TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TUOMAS KOTILAINEN

SOVELLUKSEN KEHITYS HISTORIADATAN HYÖDYNTÄMISEEN PROSESSITEOLLISUUDESSA

Diplomityö

Tarkastaja: professori Seppo Tikkanen Tarkastaja ja aihe hyväksytty 27. syyskuuta 2017

TIIVISTELMÄ

Tuomas Kotilainen: Sovelluksen kehitys historiadatan hyödyntämiseen prosessiteollisuudessa
Tampereen teknillinen yliopisto
Diplomityö, 53 sivua, 15 liitesivua
Syyskuu 2017
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Fluid Power Automation
Tarkastaja: professori Seppo Tikkanen
Avainsanat: prosessiteollisuus, historiadata, verkkosovellus

Työn tavoite oli luoda sovellus suomalaisen muovikalvon valmistajan tuotantolaitokseen, jolla voidaan analysoida linjastojen historiadataa sekä seurata prosesseja reaaliajassa. Sovellus toimii myös pohjana muissa teollisuudenaloissa käytettävälle historiadataa hyödyntävälle analyysisovellukselle.

Työn suorittamista varten perehdyttiin historiadatan hyödyntämiseen yleisesti ja erityisesti prosessiteollisuudessa, sekä erilaisiin mahdollisiin kehitystapoihin sovelluksen luomista varten.

Työn lopputuloksena on asiakkaan ja käyttökohteen vaatimuksia vastaava verkkosovellus, jolla asiakas pystyy luomaan haluamansalaisia kuvaajia mittausdatasta, laskemaan prosessin valvonnassa käytettäviä vertailuarvoja sekä hallinnoimaan sovelluksen eri määrityksiä.

Työssä lisäksi pohdittiin työn laajuuden ulkopuolelle jääneitä asioita, kuten varsinaista historiadatan keräystä sekä ohjelmiston mahdollista jatkokehitystä.

ABSTRACT

Tuomas Kotilainen: Developing software for utilizing history data in process industry
Tampere University of Technology
Master of Science Thesis, 53 pages, 15 Appendix pages
September 2017
Master's Degree Programme in Automation Technology
Major: Fluid Power Automation
Examiner: Professor Seppo Tikkanen
Keywords: process industry, history data, web application

The goal of the work was to create a software for a Finnish plastic film manufacturer that can be used to analyze the history data gathered from the production line and to monitor the processes in real time. The software also serves as a basis for history data utilizing analysis software for other fields of industry.

In order to complete the work, I familiarized myself with examples of utilizing history data in general and more specifically in process industry. I also researched potential development methods for creating the software.

The end result of the work is a web application that meets the requirements set by the client and by the use environment. The client can use the software to produce various graphs, to calculate reference values used in real time monitoring and to administrate the various definitions of the software.

In the work I also briefly covered topics beyond the initial scope such as the actual gathering of history data and the potential future development of the software.

ALKUSANAT

Tämä diplomityö on laadittu Tampereen teknilliselle yliopistolle vuoden 2017 syksyllä. Työn tekijä haluaa kiittää kaikkia työhön liittyneitä TTY:n ja Denovo Oy:n työntekijöitä.

Tampereella, 27.9.2017

Tuomas Kotilainen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

GPS	Global Positioning System
OLE	Object Linking and Embedding
IoT	Internet of things, esineiden internet
PLC	Programmable logic controller, ohjelmoitava logiikka
OPC	OLE for Process Control, teollisuustelekommunikaation standardeja ja määritelmiä
SQL	Structured Query Language, relaatiotietokantojen kyselykieli
OleDb	Object Linking and Embedding Database
ASP.NET	Active Server Pages .NET viitekehyksessä, dynaamisten internetsivun luomiseen käytetty ohjelmointimenetelmä
C#	C sharp, ohjelmointikieli
HTML	Hypertext Markup Language Hypertekstin merkintäkieli, internetsivun kirjoittamiseen käytetty kuvauskieli
WebGl	Web Graphics Library, selaimen grafiikkakirjasto
SVG	Scalable Vector Graphics, vektorigrafiikkaformaatti
3D	kolmiulotteinen
DAO	Data Access Object
DAL	Data Access Layer
LINQ	Language Integrated Query
JSON	JavaScript Object Notation

1. JOHDANTO

Historiadatan hyödyntäminen prosessien parantamiseksi ja tarkkailemiseksi on oleellista lähes jokaisella teollisuuden alalla. Pelkästään aiemmista projekteista kerätyn kokemuksen käyttämisen voi mieltää eräänlaiseksi historiadatan hyödyntämiseksi.

Kuitenkin prosessiteollisuudessa prosessista saatujen mittausten tehokkaaseen hyödyntämiseen tarvitaan avuksi tietoteknisiä sovelluksia.

Tämä työ tehtiin Denovo Oy:n palveluksessa ja työn asiakkaana on suomalainen muoviteollisuuden yritys, joka tarvitsi automatisoituihin muovinvalmistuslinjastoonsa verkkosovelluksen. Sovellusta tarvittiin linjastojen historiadatan hyödyntämiseen analysoinnissa ja monitoroinnissa.

Työn tavoite oli luoda sovellus, joka täyttää asiakkaan esittämät vaatimukset, sekä toimii pohjana muissa teollisuudenaloissa käytettävälle historiadataa hyödyntävälle analyysisovellukselle.

Työn suorittamista varten perehdyttiin historiadatan hyödyntämiseen yleisesti ja erityisesti prosessiteollisuudessa, sekä erilaisiin mahdollisiin kehitystapoihin sovelluksen luomista varten.

Työn lopputuloksena on asiakkaan ja käyttökohteen vaatimuksia vastaava verkkosovellus, sekä mahdolliset kehityksen aikana havaitut sovelluksen potentiaaliset uudet kehityssuunnat.

Työn luvussa 2 on käsitelty historiadatan hyödyntämistä, sen tarjoamia mahdollisuuksia ja sovelluksen käyttöympäristöä. Luvussa 3 on käsitelty työssä sovelluksen kehitykseen tarjolla olleita teknologiavaihtoehtoja sekä käytettäväksi valittuja teknologioita ja tehtyihin valintoihin johtaneita syitä. Luvussa 4 on esitelty sovelluksen kehityksen eri osa-alueet ja niiden toteutukset ja luvussa 5 on pohdittu sovelluksen mahdollisia tulevia kehityssuuntia.

2. HISTORIADATAN HYÖDYNTÄMINEN

Historiadataa voidaan hyödyntää monilla eri tieteenaloilla. Esimerkiksi kauppa-alalla on tutkittu jälleenmyyntiliikkeiden asiakkaita ja heidän tekemistään liiketapahtumista muodostunutta historiadataa, jonka avulla asiakkaat on voitu asettaa luottoriskin mukaan ryhmiin heille datasta muodostetun riskitason perusteella [1].

Matkailualalla on kehitetty mobiilisovellus, jonka tarkoitus on hyödyntää käyttäjän historiadataa. Sovellus vertaa käyttäjän Global Positioning System (GPS) paikkatietojen sekä aktiviteettitietojen muodostamaa historiadataa ja suosittelee sen perusteella käyttäjälle uusia aktiviteettejä [2].

Myös monessa prosessiteollisuuden alueessa voidaan hyödyntää aikaisemmista prosesseista kerättyä dataa. Esimerkiksi monen jäähdyttimen järjestelmässä on tarkasteltu historiadataa ja sen perusteella on tunnistettu mahdollisuuksia parantaa jäähdyttimien suoritusta. [3].

Sensoriverkoissa historiadataa on hyödynnetty paikkatietojen varmentamisessa järjestelmässä, jossa sensorit tuottavat paikkakohtaisia mittauksia. Järjestelmän verkossa olevilta sensoreilta kerätään historiadataa, jota analysoimalla voidaan uusille mittauksille muodostaa ns. uskottavuus. Uskottavuuden perusteella voidaan varmentaa sensorin paikkatieto ja havaita virheelliset arvot [4].

Historiadatan analysointia käytetään myös teollisuudessa tuotantolaitteiden huoltamiseen. Teollisuuden toimilaitteiden huoltoraporteista muodostettua historiadataa analysoimalla tähän tarkoitukseen räätälöidyillä tiedonlouhinta-algoritmeilla on voitu optimoida laitteiden huoltotoimenpiteitä [5].

Koska kehitettävän sovelluksen käyttökohteessa, eli muovinvalmistuslinjastossa, mitattujen arvojen halutaan pysyvän tiettyä tuotetta valmistettaessa tietyissä rajoissa, on sovelluksen järkevä hyödyntää historiadataa, joka muodostettaisiin aiemmista vastaavan tuotteen valmistuksesta saaduista mittauksista.

Jos aiemmista valmistuseristä voidaan erottaa ajanjaksoja, jolloin tuotteen laatu on ollut hyvä ja siten myös linjaston mitatut arvot ovat olleet halutuilla tasoilla, voidaan ajanjakson historiadatasta tarkastella mittausten keskiarvoja ja hajontoja ja verrata niitä uutta erää valmistettaessa otettaviin mittauksiin.

Historiadatasta saatu hyöty riippuu siis tässä käyttötapauksessa paljolti käyttäjän kyvystä valita sopivia jaksoja historiadatasta.

Tässä sovelluskohteessa historiadataa hyödynnetään vain prosessin seurantaan ja valvontaan, mutta yhdistämällä historiadatan käyttö simulaatioihin voitaisiin historiadataa käyttää esimerkiksi ennakoivan säädön parantamiseen.

Jos käyttökohteena olevasta linjastosta olisi olemassa hyviä simulaatioita, voitaisiin ennen sovelluksen käyttöönottoa muodostaa simuloitua historiadataa ja sovelluksen valvovia toiminnallisuuksia voitaisiin käyttää heti käyttöönotosta asti. Kuitenkin tässä tapauksessa jokaista tuotetta, jolle halutaan historiadatasta vertailuarvot, joudutaan valmistamaan ainakin yksi erä.

Sovellusta kehitettäessä on yritetty huomioida mahdollinen jatkokehitys muitakin prosessiteollisuuden aloja silmällä pitäen. Historiadataa voidaan hyödyntää minkä vain prosessin kanssa, joka halutaan suorittaa useampaan kertaan, mutta muissa käyttökohteissa voidaan tarvita monimutkaisempaa analyysiä kuin vain arvojen vertailua historiadatan keskiarvoihin ja hajontoihin.

2.1 Big Data

Big Datalla tarkoitetaan datajoukkoja joita ei voida hallita yleisillä tietokoneilla tai perinteisillä tietokanta ohjelmistoilla [6]. Vaikka tämä historiadatan hyödyntämistapaus putoaa Big Datan määritelmän ulkopuolelle molemmissa kriteereissä, on soveltamisessa hyödynnetty joitain Big Datan keskeisiä ajatuksia

Sovelluksen suunnittelun keskiössä on ollut ajatus kerätä prosessista kaikki mahdollinen mittausdata, jota voidaan myöhemmin hyödyntää prosessin seurantaan ja parannukseen.

Big Data pyrkii lähtökohtaisesti hyödyntämään erittäin suuria datamääriä. Kuitenkaan sovelluskohteesta saatava datan määrä ei ole verrattavissa yleisiin Big Datan sovelluksiin. Big Datassa tieto on myös usein monimuotoisempaa ja rakenteettomampaa kuin tässä sovelluskohteessa.

Osaltaan suuren mittausdatamäärän keräämisen mahdollistaa esineiden internetin yleistyminen teollisuudessa.

2.2 Esineiden internet

Esineiden internetillä (Internet of things, IoT) tarkoitettaan yleensä arkipäiväisten esineiden yhteen liittämistä tietoverkkojen avulla [7]. Yleisimpiä esimerkkejä ovat erilaiset älylaitteet kuten älytelevisiot ja -jääkaapit. Esineiden internetillä on myös sovelluksia teollisuuskäytössä.

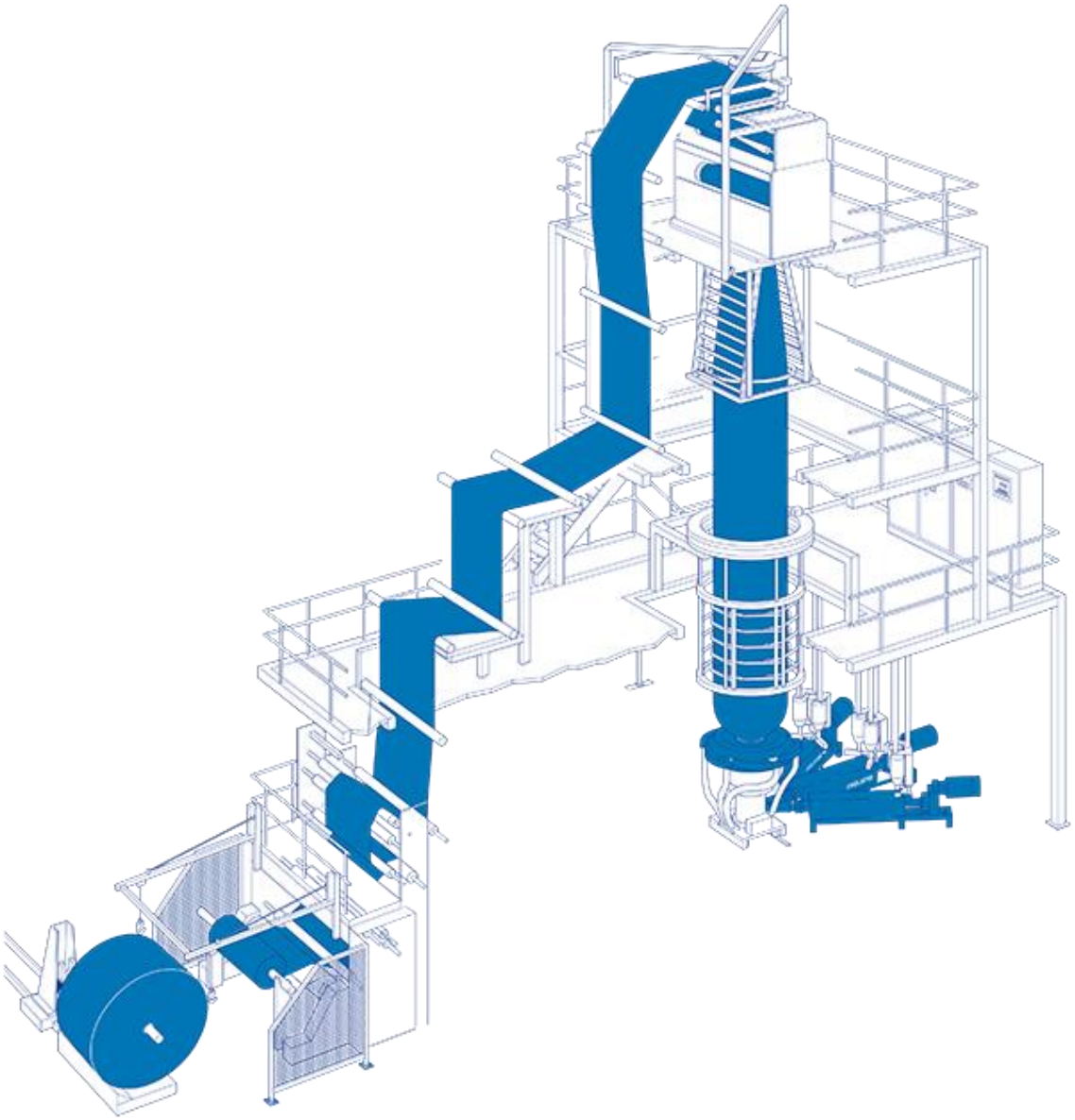
Sovelluksen toteutusympäristöön kuuluu yksi ohjelmoitava logiikka (Programmable logic controller, PLC), joka voidaan kytkeä laitoksen tietoverkkoon ja siten sen voidaan katsoa kuuluvan teolliseen esineiden internettiin.

Esineiden internet mahdollistaa entistä helpomman tiedonvälityksen, joka osaltaan helpottaa suuresti sovelluksen tarvitsemaa historiadatan keräystä.

Toisaalta ulospäin yksinkertaisemmalta vaikuttava tiedonsiirto ei ole välttämättä laskennallisesti yhtä tehokasta kuin esimerkiksi sarjaliikenne, ja tietoverkkoihin yhdistyminen tuo mukanaan aina mahdollisia tietoturva-aukkoja.

2.3 Toteutusympäristö

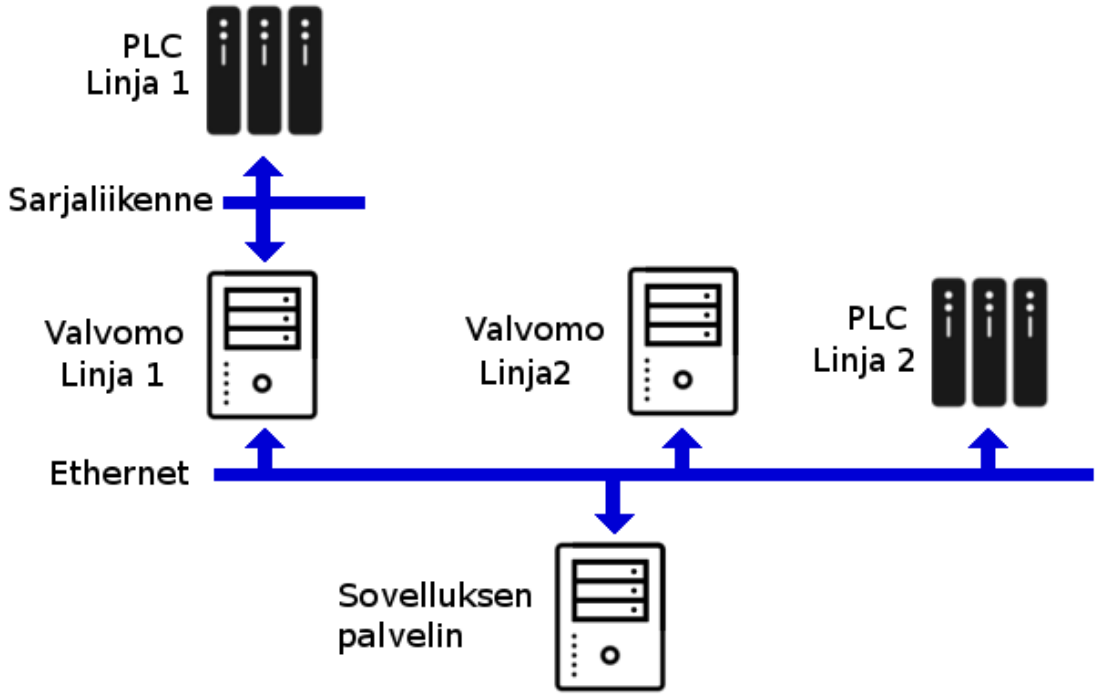
Muovinvalmistuslaitoksessa, johon sovellus haluttiin käyttöön, on kaksi linjastoa. Molemmilla linjastoilla on oma(t) logiikkansa sekä valvomotietokone. Järjestelmään kuuluu myös niin sanottu Data Logger. Kuvassa Kuva 1 on esimerkki eräästä laitoksessa käytetystä kalvonpuhalluslinjastosta.



Kuva 1 Kalvonpuhalluslinja [8]

Data Loggerilla tarkoitetaan tässä sovellusta, joka tallentaa linjastosta saatavat mittausdatat historiadataan. Tässä tapauksessa sovellukseen on lisätty taustaprosessi, joka lukee linjastojen mittausarvoja ja tallentaa niitä tietokantoihin.

Sovellus ja siihen liittyvä Data Logger haluttiin käyttöön omalle tietokoneelleen, joka on laitoksen verkossa. Toisessa linjastossa on käytössä uudempi Mitsubishin Q-sarjan logiikka, joka on suoraan samassa verkossa sovelluksen tietokoneen kanssa, mutta toisessa linjastossa on käytössä vanhempi A-sarjan logiikka, joka ei ole kytketty verkkoon. Ainoa yhteys toiseen logiikkaan on siis kyseisen linjaston valvomotietokoneen kautta.



Kuva 2 Laitoksen verkko

Laitoksen laitteiden kytkeytyminen verkkoon on kuvattu kuvassa Kuva 2. Suoraan verkossa kiinni olevasta logiikasta saadaan data suoraan sovelluksen palvelinkoneelle käyttämällä siihen asennettua Object Linking and Embedding for Process Control (OLE for Process Control tai OPC) palvelinta, jolta Data Logger saa datan ja kirjoittaa ne sitten sovelluksen käyttämään tietokantaan.

Toisenkin logiikan data voitaisiin myös kerätä OPC palvelimella, sillä OPC palvelin ei välttämättä vaadi toimiakseen Ethernet yhteyttä logiikkaan. Riippuen käytetystä OPC palvelinsovelluksesta voi sovellus tarjota ajurit, jotka tarvitaan logiikan datan saamiseksi sarjaliikenteen kautta.

Asiakas ei kuitenkaan halunnut asentaa valvomotietokoneille uusia ohjelmistoja ja valvomojen ikä huomioon ottaen se on todennäköisesti paras ratkaisu. A-sarjan logiikalta datan saamiseen tarvittiin siis eri ratkaisu.

Valvomo tietokoneissa on käytössä Wonderware InTouch, jota käytetään linjaston arvojen tarkkailuun. Wonderware on Schneider Electricin kehittämä teollisuusohjelmisto, joka tarjoaa mm. käyttöliittymä-, toiminnanohjaus-, tuotannonohjaus- sekä valvomotoiminnallisuuksia. InTouch on Wonderwaren käyttöliittymätyökalu.

InTouch tarjoaa mahdollisuuden ajaa koodia ohjelman yhteydessä. InTouchin käyttämä script-kieli on erittäin yksinkertainen ja rajoittunut, eikä tarjoa esimerkiksi silmukkarakenteita. Puutteista huolimatta sitä voidaan käyttää datan kirjoittamiseen.

InTouchin koodilla voidaan kirjoittaa mittauksien arvot tekstitiedostoon, joka on jaetussa kansiossa, jolloin Data Logger voidaan muokata lukemaan mittausdatat Ethernetin yli tekstitiedostosta.

Koska mittauksia on useita satoja, ja jokaisen mittauksen kirjoittamiselle tarvitaan koodin rajoituksien takia oma rivi, on kirjoittavan koodin ylläpito erittäin vaivalloista. InTouch ei myöskään tue kuin tiettyä määrää komentoja taustalla suoritettavassa koodissa. Tämä johtuu oletettavasti siitä, että InTouch suorittaa taustalla koodia tietyn asetetun millisekuntimäärän välein, ja jos kaikkia rivejä ei ehditä suorittaa ennen uutta sykliä, sovellus todennäköisesti kaatuu. Käyttökohteessa mittauksia halutaan tallentaa vain noin viiden minuutin välein, mutta taustakoodia ajetaan kerran neljässä sekunnissa. Ongelmaa ei olisi, jos InTouch tarjoaisi mahdollisuuden ajaa useampaa taustakoodia (jotka olisivat oletettavasti omissa säikeissään) eri ajastimilla.

Sovelluksen käyttäjinä on muovinvalmistuslinjaston työntekijöitä, joilla ei oleteta olevan erityistä tietoteknistä osaamista. Sovellus on siis yritetty tehdä mahdollisimman helppokäyttöiseksi ja käyttäjävirheitä sietäväksi.

3. TEKNOLOGIAVAIHTOEHDOT JA -VALINNAT

Sovelluksen eri osa-alueiden toteutuksiin on tarjolla monia eri teknologiavaihtoehtoja. Joidenkin osa-alueiden kohdalla projektin johto oli määritellyt käytettävän teknologian ja toisten kohdalla olin itse vapaa valitsemaan haluamani teknologian.

3.1 Tietokannat

Projektin johto oli ennen sovelluksen toteutusta määritellyt, että sovellus kehitetään käyttämään Microsoft Access-tietokantoja. Tässä kappaleessa on kuitenkin myös pohdittu Structured Query Language (SQL) tietokantojen tuomia mahdollisia hyötyjä.

Access on Microsoftin kehittämä tietokantojen hallintajärjestelmä, joka yhdistää relationaalisen Microsoft Jet Database Enginen ja graafisen käyttöliittymän.

Accessin tarjoama graafinen käyttöliittymä on hyödyllinen sovelluksen kehitysvaiheessa nopeaan ja vaivattomaan tietokannan muutoksien tarkasteluun, mutta sovelluksen käyttöönoton jälkeen käyttöliittymää ei oletusarvoisesti enää tarvita.

Kun sovellusta kehitetään käyttämään Access-tietokantoja, joudutaan sovelluksessa käyttämään Object Linking and Embedding Database (OleDb) rajapintaa. OleDb:n käyttämisen etuna on, että samaa rajapintaa voidaan käyttää myös muiden tietokantojen kanssa, jos tulevaisuudessa halutaan siirtyä pois Access-tietokantojen käytöstä.

SQL:a varten on olemassa oma OleDb:n toiminnallisuuksia vastaava rajapinta, joka lisäksi tarjoaa tiettyjä etu OleDb:en verrattuna. SQL tukee nimettyjä parametrejä, kun taas OleDb:n kanssa parametrit tunnistetaan niiden järjestyksestä. Lisäksi SQL:n ajureiden mainostetaan olevan nopeampia.

Microsoft Jet Database Enginessä on myös perustavanlaatuinen ongelma, jos käytettävät tietokannan taulut ovat suuria. Jet Engine noutaa kyselyllä tietokannasta koko taulun asiakkaan muistiin ja seuloo vasta sitten taulusta halutut kentät, kun taas SQL-palvelin pystyy hoitamaan seulonnan palvelimen puolella [9]. Sen lisäksi, että tämä kuluttaa isojen taulujen kanssa paljon asiakaspuolen resursseja, aiheuttaa tämä myös suurempaa liikennettä verkon yli.

Access-tietokantojen kanssa voi myös aiheutua ongelmien useiden käyttäjien yrittäessä samanaikaisesti käyttää samaa tietokantaa [9]. Tämä aiheutuu tavasta, jolla Access lukit-

see tietokannan kenttiä. SQL hallitsee kenttien lukitsemisen yleisesti paremmin. Sovelluksen nykyisessä käyttökohteessa on oletusarvoisesti vain yksi käyttäjä, joten tässä tapauksessa ongelma ei vaadi huomiota.

3.2 Verkkosovelluksen viitekehys

Koska sovelluksen käyttöliittymä haluttiin selaimesta käytettäväksi, sovelluksen kehitystä lähestyttiin ajatuksella tehdä sovelluksesta palvelimella toimiva verkkosovellus. Verkkosovelluksen kehitykseen haluttiin valita sopiva viitekehys.

ASP.NET on Microsoftin kehittämä verkkosovellusten viitekehys. Lyhenne ASP tulee sanoista Active Server Pages. ASP.NET käyttää nimensä mukaisesti .NET sovellus viitekehystä, joka on niin ikään Microsoftin kehittämä.

Viitekehystä päädyttiin käyttämään, koska .NET viitekehys on yhteensopiva monien ohjelmointikielien kanssa ja se tarjoaa valmiita alemman tason toiminnallisuuksia, kuten muistin hallintaa. Yksi .NET viitekehyksen tukemista ohjelmointi kielistä on C sharp (C#), jota haluttiin käyttää sovelluksen kehityksessä.

Käytettäessä ASP.NET viitekehystä, voidaan valita kolmesta eri versiosta: Web Forms, Model View Control ja Web Pages. Versioiden välillä sovelluksen kehitys on erilaista, mutta kaikki versiot tarjoavat kaikki viitekehyksen toiminnallisuudet [10].

Web Formsin käyttö vaatii pikemminkin C# ohjelmointikielen tuntemusta kuin Hypertext Markup Languagen (HTML) osaamista, mikä oli yksi sen käyttöön valintaan johtaneista seikoista.

ASP.NET Web Formissa sivut on jaettu .aspx koodiin, joka sisältää näytettävän merkinnän ja .aspx.cs koodiin, joka sisältää varsinaisen prosessoinnin. Yhdessä sivut muodostavat koodin ja merkinnän, joka tarvitaan selaimessa näytettävän HTML:n luomiseen [11].

ASP.NET viitekehyksen etu on sen elinkaaren tarjoama mahdollisuus käyttää informaatiota HTML:n luomisen eri vaiheissa. Tämän kääntöpuolena on kuitenkin elinkaaren tuoma kehityksen monimutkaistuminen, sillä sama koodi saattaa tuottaa eri tuloksia riippuen missä elinkaareen vaiheessa sitä kutsutaan [11].

Web Forms tarjoaa myös sovelluksen kehitystä nopeuttavia Controlleja. Controllit ovat objekteja, jotka muistuttavat HTML-elementtejä mutta ne voivat sisältää monimutkaisiakin toiminnallisuuksia. Valmiiden Controllien lisäksi kehittäjä voi käyttää itse tekemiään User Controlleja [11].

3.3 Kuvaajat

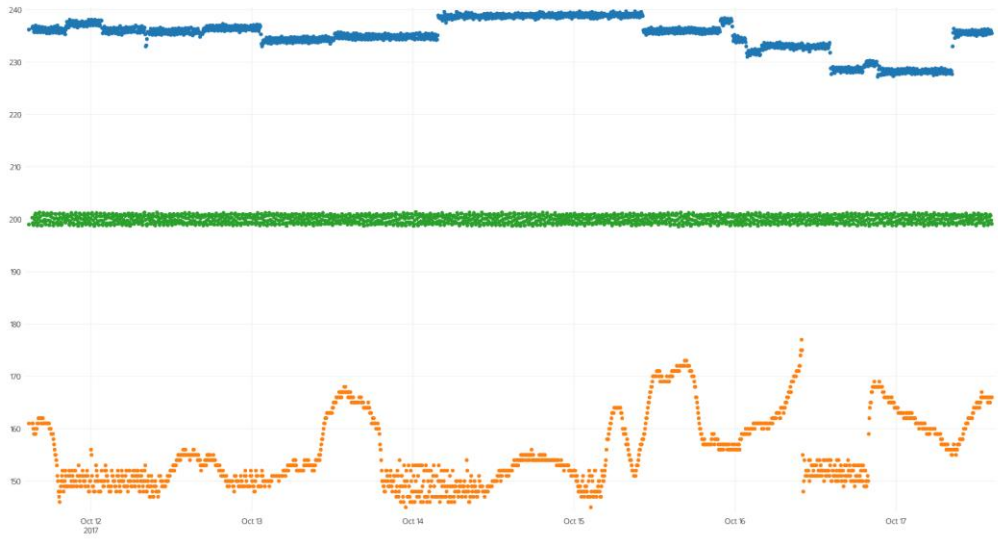
Sovellukseen haluttiin mahdollisuus muodostaa kuvaajia ja koska sovelluksen käyttöliittymä toimii selaimessa, täytyy se ottaa huomioon kuvaajien piirtoon tarkoitettua teknologiaa valittaessa.

Käytettäväksi kirjastoksi päädyttiin valitsemaan plotly.js, joka on JavaScriptiä käyttävä versio Plotlystä. Tämä tarkoittaa, että kirjastoa voidaan käyttää verkkosivussa kuvaajien piirtämiseen.

Plotlyn JavaScript versio on ilmainen ja avoimeen lähdekoodiin perustuva [12] ja sitä pidetään yleisesti nopeana ja tehokkaana kuvaajakirjastona. Kirjaston luomat kuvaajat ovat myös dynaamisia, mikä sopi hyvin tähän kehitystarkoitukseen.

Plotly.js on rakennettu d3.js ja stack.gl päälle. Data-Driven Documents eli d3.js on JavaScript kirjasto, jota käytetään dataan perustuvien dokumenttien manipuloimiseen [13]. Stack.gl on avoin sovellus ekosysteemi Web Graphics Librarylle (WebGl) [14].

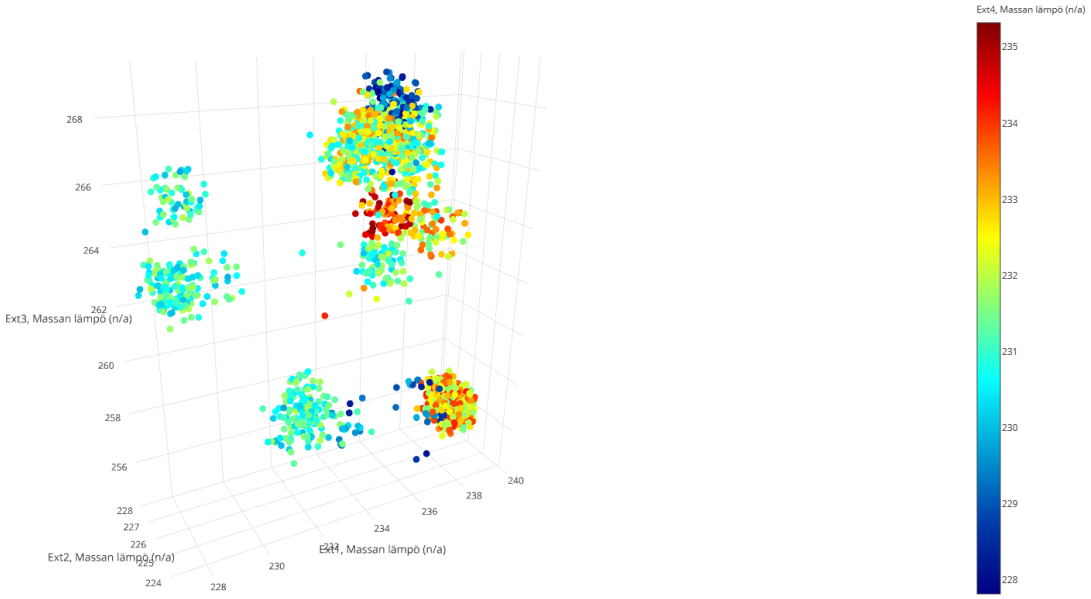
WebGl lisäksi kuvaajien piirtämiseen voidaan käyttää Scalable Vector Graphicsiä (SVG), mutta tässä käyttötapauksessa WebGl on parempi vaihtoehto, sillä WebGl tukee paremmin suurien datapistemäärien piirtämistä.



Kuva 3 esimerkki Plotly.js kuvaajasta

Kuvassa Kuva 3 on esitetty esimerkki kuvaajasta, jossa on kolmen mittauksen arvot viiden minuutin välein noin kuuden päivän pituiselta jaksolta. Datapisteiden määrästä voidaan nähdä miksi kuvaajien piirtämiseen täytyy käyttää WebGl:ää.

Plotly.js voidaan käyttää myös kolmiulotteisten (3D) kuvaajien muodostamiseen, joka oli yksi asiakkaan sovellukselle asettamista vaatimuksista.



Kuva 4 3D kuvaaja

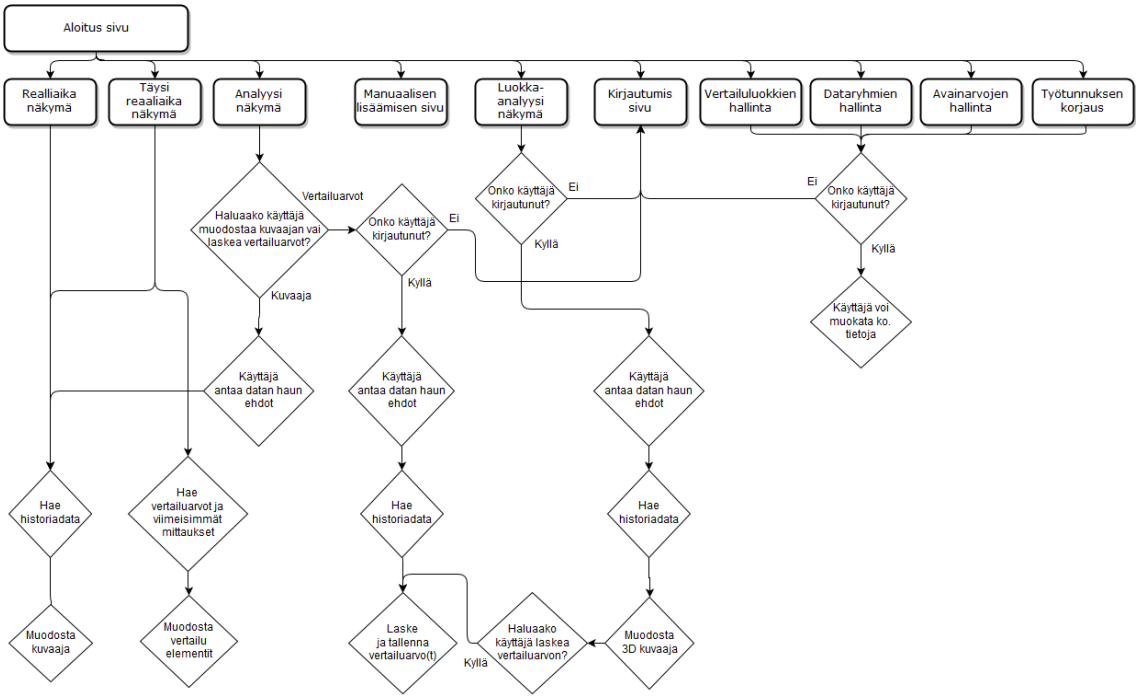
Kuvassa Kuva 4 on esimerkki kuvaajasta, jossa kolme mittausta muodostaa 3D kuvaajan akselit ja neljäs mittaus on esitetty datapisteiden väriskaalana.

4. SOVELLUKSEN TOTEUTUS

Sovellus toteutettiin tässä osiossa kuvatulla tavalla. Sovelluksen eri osa-alueet on eritelty omien otsikoidensa alle.

Sovellus toteutettiin selaimesta käytettäväksi, ja osa-alue kohtaisissa otsikoissa on selitetty mahdollisimman suppeasti, miten käyttöliittymä on toteutettu ja keskitytty enemmän selittämään sovelluksen toiminnallisuutta eri osa-alueissa. Käyttöliittymän elementtejä on mainittu tarpeen mukaan, mutta tarkempaa tieto käyttöliittymän toteutuksesta ja käyttäjäkokemuksen suunnittelusta on luvussa 4.11 Käyttäjäkokemus.

Sovellus käyttää ASP.NET viitekehystä ja Microsoft Access-tietokantoja, joita on yksi yhteinen kaikille sovelluksen toiminnoille sekä yksi erillinen tietokanta jokaista linjastoa kohden. Sovelluksen nykyisessä käyttökohteessa on kaksi linjastoa, eli käytössä on kolme Access-tietokantaa.



Kuva 5 Sovelluksen vuokaavio

Kuvassa Kuva 5 on esitetty karkeasti ohjelmiston toiminnallisuus vuokaaviolla.

4.1 Viitekehys

Valitsin sovelluksen toteutuksessa käytettäväksi viitekehykseksi ASP.NET viitekehyksen, sillä ASP.NET:n palvelinpuolen koodissa käytetään C# ohjelmointikieltä, joka oli minulle entuudestaan tutuin. Lisäksi projektin johtaja ehdotti toteutukseen käytettävän juuri C#.

Kuitenkaan tässä sovelluksen kehitykseen valittu viitekehys ei ole kovin tärkeä valinta, sillä sovellus toimii itsenäisesti eikä sitä tarvitse esimerkiksi integroida toimimaan osana toista palvelinsovellusta.

ASP.NET:ssä asiakaspuolen koodissa voidaan määritellä ns. Update Panel elementtejä. Näiden elementtien sisällä olevat muut asiakaspuolen elementit päivittävät tilojaan ja arvojaan ilman että koko sivu täytyy ladata uudestaan. Suurin osa sovelluksen sivuista on kiedottu Update Paneliin, sillä monet käyttöliittymissä tehdyt valinnat vaikuttavat muihin käyttöliittymän elementteihin.

ASP.NET tarjoaa mahdollisuuden käyttää ns. User Controlleja, joiden avulla sovelluksen eri sivuilla voidaan viitata samaan koodiin. Tämä on erittäin hyödyllinen ominaisuus, jos sivuston eri sivuilla käytetään samoja elementtikokonaisuuksia, kuten tämän sovelluksen toteutuksessa on käytetty.

Toteutus aloitettiin ns. analyysinäkymän kehittämisestä, sillä työnantajan antamien tietojen perusteella se olisi helpoin toteuttaa aluksi ja samalla voisin totuttautua ASP.Net viitekehyksen käyttöön.

4.2 Analyysinäkymä

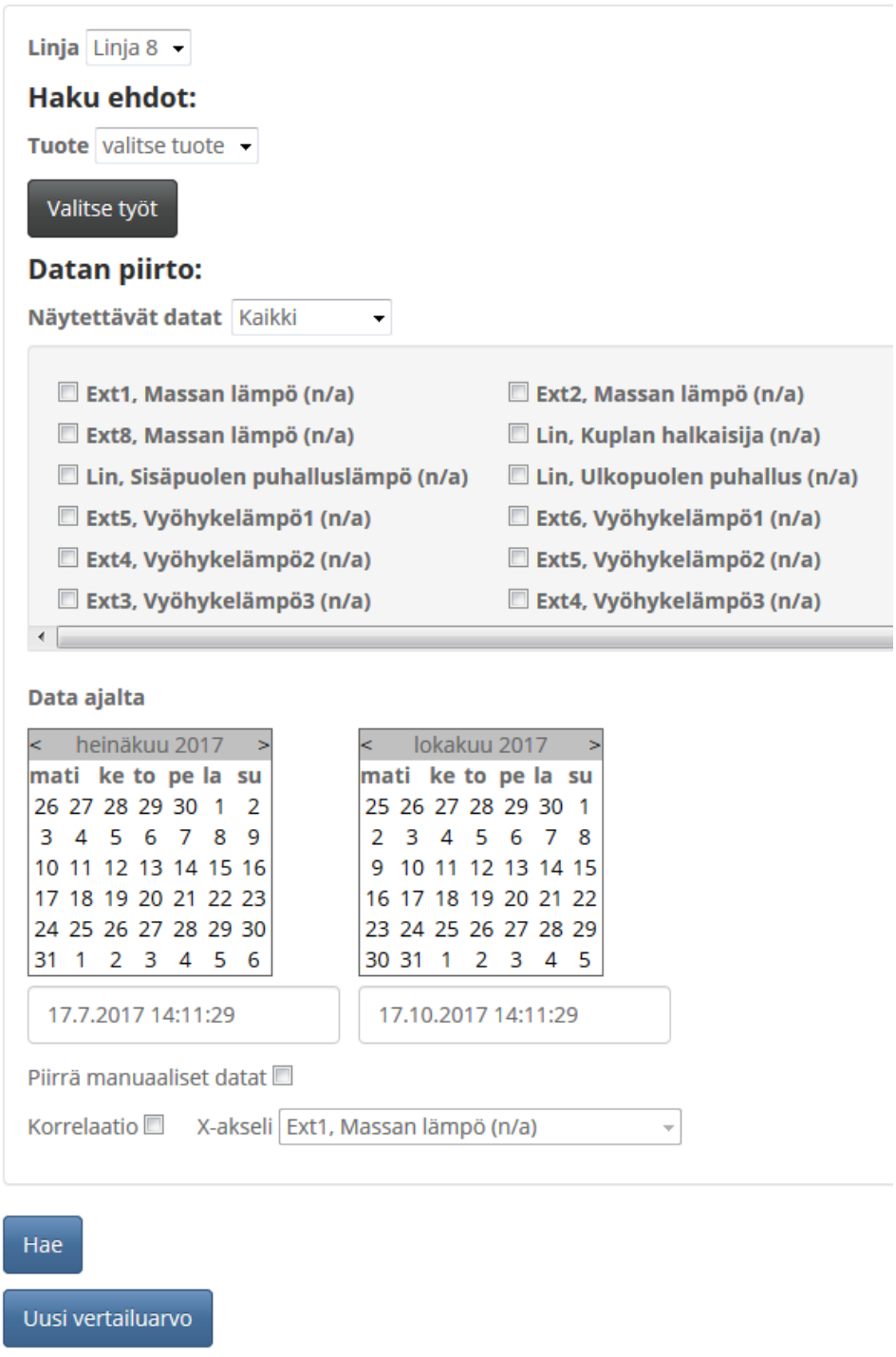
Analyysinnäkymän perusidea oli tarjota käyttöliittymä, jolla voidaan piirtää kuvaaja tietyillä ehdoilla valitusta historiadatasta.

Analyysinäkymään haluttiin mahdollisuus valita kuvaajaan piirrettävä mittausdata halutulle tuotteelle ja työlle halutun linjaston historiadatasta. Lisäksi mittausdatan piirtoon haluttiin mahdollisuus rajata valittu historiadata ajan mukaan.

ASP.NET viitekehys tarjoaa valmiita käyttöliittymäelementtejä kuten pudotusvalikkoja, valintaruutuja ja valintaruutulistoja, tekstikenttiä sekä kalentereita.

Käyttöliittymä koostuu ns. lomakkeesta, joka on kiedottu Update Paneliin ja ”Hae”-painikkeesta, jolla kuvaaja lopulta muodostetaan.

Lisäksi ylläpitäjän oikeuksia käyttävälle käyttäjälle näytetään ”Uusi vertailuarvo”-painike, jolla voidaan laskea ja tallentaa uudet vertailuarvot tietokantaan valittujen ehtojen mukaisesti. Vertailuarvoista on kirjoitettu tarkemmin luvussa 4.5 Vertailuarvot.



Kuva 6 Analyysinäkymä

Kuvassa Kuva 6 on esitetty analyysinäkymän lopullinen versio, jonka eri toiminnallisuudet on selitetty tämän kappaleen alaosioissa.

4.2.1 Kuvaajaan piirrettävien mittauksien valinta

Eri mittauksien valittavat historiadatat esitettään valintaruutulistalla, johon voidaan sitoa haluttujen rajausten mukaisesti tietokannan ”IO_DEF” taulukossa määriteltyjä mittauksia.

Sovellus siis hakee sivun latauksessa ”IO_DEF” taulukosta haluttujen mittauksien ID:t ja kuvaukset. Haussa käytetään OleDb kyselyä, joka palauttaa datapöydän jonka sarakkeet ovat ID ja kuvaus ja joka sisältää halutut mittaukset.

Ohjelma 1:ssa esitetyssä kyselyssä sovellus hakee ”DByhteys” luokan ”HaeTauluNoPar” metodia käyttämällä halutut mittaukset datatauluun ”arvot”. ”DByhteys” luokan toimintaa on tarkemmin selitetty luvussa 4.4 Data Access Layer.

Yksinkertaisesti selitettynä käytetty metodi ottaa kaksi argumenttia joista ensimmäinen on varsinainen OleDb kysely ja toinen on ns. Connection String, joka määrittää mihin tietokantaan kysely lähetetään. Connection Stringeistä on tarkempaa tietoa luvussa 4.2.7 Linjaston valinta ja luvussa 4.10.1 Web.config.

ASP.NET viitekehys mahdollistaa datan sitomisen sen tarjoamiin elementteihin. Käytännössä tämä tarkoittaa, että valintaruutulistaan luodaan jokaista OleDb kyselyn palauttaman datataulun riviä kohden yksi valintaruutu, jolla on nimikkeenä haettu kuvaus ja arvona haettu ID.

Ohjelma 2:ssa on esitelty esimerkki, missä ”arvot” on kyselyn palauttama taulu, josta sidotaan kentät ”ID” valintaruutulistan ”IOlista” valintaruutujen arvoiksi ja kentät ”Description” valintaruutujen nimikkeiksi.

Dataa elementtiin sidottaessa elementistä poistetaan listaesineet ”.Clear()” metodilla. Tämä ei ole kaikissa tapauksissa välttämätöntä, sillä uudessa datan sitomisessa vanha datasidonta korvataan, mutta jos elementtiin lisätään manuaalisesti listaesineitä, jää manuaalisesti lisätty listaesine elementtiin mistä aiheutuu turhia kaksoiskappaleita.

Nyt voidaan siis kutsua valintaruutulistaa ja esimerkiksi tarkistaa mitkä valintaruudut siitä on valittuna ja mitkä niiden arvot ovat. Näin voidaan myöhemmin muodostaa kysely, jolla haetaan asianmukaiset historiadatat kuvaajaa varten.

Sovellus hakee valintaruutulistaan oletusarvoisesti kaikki ”IO_DEF” taulukon mittaukset. Tällainen näkymä voi kuitenkin olla joissa tapauksissa aivan liian sekava, joten sovellukseen tarvitaan mahdollisuuksia järjestellä mittauksia pienempiin ryhmiin tai joukkoihin.

4.2.2 Näytettävien mittauksien valinta

Analyysinäkymään päätettiin lisätä kaksi tapaa seuloa mittauksia.

Seulonta mittaustyypin mukaan

Seulonnalla mittaustyypin mukaan tarkoitettaan mahdollisuutta näyttään vain yhden mittaustyypin eri ekstruudereiden mittaukset.

Jos valitaan mittaustyypin mukainen seulonta, sivulle lisätään pudotusvalikko, mihin sidotaan kaikki eri ”IO_DEF” taulukossa määritellyt mittaustyypit. Koska ”IO_DEF” taulukossa on määritelty kaikkien mittauksien kuvauksien olevan sama samantyyppisille mittauksille, voidaan pudotusvalikkoon hakea listaesineet Ohjelma 3:ssa esitetyn koodin mukaisesti.

Pudotusvalikkoon lisätään erillinen listaesine linjastolle yhteisiä mittauksia varten. Jos nyt halutaan hakea ”IO_DEF” taulukosta valintaruutulistassa näytettävät mittaukset, tehdään se Ohjelma 4:ssa esitetyn koodin mukaisesti.

Seulonta mittausryhmän mukaan

Seulonnalla mittausryhmien mukaan tarkoitetaan mahdollisuutta näyttää vain valitun mittausryhmän mittaukset. Mittausryhmät ovat järjestelmähallitsijan oikeuksilla toimivan käyttäjän lisättävissä, muokattavissa ja poistettavissa.

Samaan tapaan kuin mittaustyypin mukaan seulottaessa, jos valitaan seulonta mittausryhmien mukaan, sivulle lisätään pudotusvalikko, johon haetaan kaikki eri ryhmät, jotka on määritelty linjaston tietokannan ”Ryhmät” taulukossa. Toisin kuin mittaustyypeille, ryhmille on määritelty sekä nimi että ID, joten pudotusvalikon nimikkeisiin sidotaan haetut nimet ja arvoihin sidotaan haetut ID:t.

Valitun ryhmän mukaan mittauksien hakeminen on esitetty kohdassa Ohjelma 5.

Sovellus hakee ”Ryhmät” taulukosta rivin joka vastaa valittua ryhmän ID:tä. Haetusta rivistä voidaan ottaa ryhmään kuuluvat ID:t listaan, jota iteroimalla voidaan lisätä näytettävät mittaukset hakevaan kyselyyn parametri jokaista ID:tä kohden. Näin kysely palauttaa ”IO_DEF” taulukosta halutut mittaukset.

Mittausryhmistä on tarkempaa tietoa luvussa 4.7.1 Ryhmien hallinta.

Analyysinäkymään siis lisättiin pudotusvalikko, jonka listaesineiden kuvaukset ovat:

1. Kaikki
2. Mittaukset
3. Ryhmät

Kun käyttäjä vaihtaa pudotusvalikossa valittua listaesinettä sovellus päivittää Update Panelin, jolloin valintaruutulista päivitetään hakemalla ja sitomalla tietokannasta halutut mittaukset.

Jos käyttöön on valittu jompikumpi seulonta, sivulla näytetään toinen pudotusvalikko, josta voidaan valita haluttu mittaustyyppi tai -ryhmä seulonta tavasta riippuen. Samaan tapaan kuin seulontaa valittaessa, kun käyttäjä vaihtaa ryhmän tai tyypin pudotusvalikon valittua listaesinettä haetaan valintaruutulistaa varten mittauksien ID:t ja kuvaukset uudestaan tietokannasta.

4.2.3 Tarkemman tiedon lisääminen mittauksien kuvauksiin

Sovelluksen muut toiminnot edellyttävät, että tietokannas ”IO_DEF” taulukossa kaikilla saman tyyppisillä mittauksilla on sama kuvaus.

Käyttöliittymän kannalta tämä tarkoittaa, että jos sivujen elementteihin sidottaisiin suoraan tietokannasta saatu kuvaus, eri ekstruudereiden samantyyppiset mittaukset näyttäisivät käyttäjälle täysin samalta.

”IO_DEF” taulukossa on kuitenkin annettu jokaiselle mittaukselle ns. Block arvo, joka kuvaa mille ekstruuderille kyseinen mittaus kuuluu.

Lisäksi taulukossa on määritelty yksiköt jokaiselle mittaukselle.

Näitä tietoja hyödyntämällä voidaan luoda iteroiva silmukan, joka lisää kyselyn palauttaman datataulun kuvauksiin etuliitteen ”ExtN, ” jossa N on extruuderin numero ja kuvauksien perään mittauksen yksikön.

Jos mittaus ei ole ekstruuderikohtainen mittaus vaan linjastolle yhteinen mittaus, sovellus lisää kuvaukseen etuliitteen ”Lin, ”.

”IO_DEF” taulukossa mittauksille on määritelty Block_Table arvo joka viittaa siihen, missä taulukossa mittauksen historiadata on tallennettuna. Siitä voidaan kuitenkin myös tarkastaa, onko mittaus ekstruuderikohtainen vai linjastolle yhteinen sen perusteella onko Block_Table arvona ”LIN” vai joku muu.

Lisäksi ns. avainarvoille (joiden Block_Table arvo on ”KEY”) on määritelty, että ne ovat linjastolle yhteisiä, jos niiden Block arvoksi on asetettu 0. Tämä johtuu siitä, että kaikki

avainarvot on määritelty niiden käsittelytavasta johtuen samassa taulukossa, jolloin linjastolle yhteisiä avainarvoja ei voida sijoittaa samaan taulukkoon kuin muita linjastolle yhteisiä mittauksia.

Silmukka on esitetty Ohjelma 6:ssa missä ”arvot” on kyselyn palauttama taulukko ja sen sarakkeet ovat Description, ID, Block, Block_Table ja Unit.

Jos ”IO_DEF” taulukossa ei ole erikseen määritelty yksikköä, sovellus lisää yksiköksi (n/a).

4.2.4 Tuotteen mukaan hakeminen

Analyysinäkymään haluttiin mahdollisuus valita, minkä tuotteen valmistuksen historiadatasta muodostetaan kuvaaja. Tähän tarkoitukseen sivulle lisättiin pudotusvalikko, johon haetaan kaikki linjaston tietokantaan tallennetut tuotetunnukset.

Linjaston tietokannassa on ”Tuotteet” taulukko, johon on koottu kaikki linjastossa käytetyt tuotetunnukset ja niitä vastaavat ID:t. Taulukossa ID kenttä on asetettu automaattiluvuksi. Tämä tarkoittaa, että taulukkoon lisättäessä uusia tuotetunnuksia tietokanta lisää uudelle tuotetunnukselle järjestyksessä seuraavan ID:n, jota ei ole aiemmin käytetty.

Automaattiluvut ottavat huomioon myös huomioon taulukosta poistetut tietueet, eli poistettujenkaan tuotetunnuksien ID:t eivät vapaudu tulevien tuotetunnusten käytettäväksi.

Poistettujen automaattilukujen käyttämättä jättäminen voi olla toivottu toiminnallisuus sovelluksissa, joissa halutaan useampaan tietokannan taulukkoon sama automaattiluku toisiaan vastaaville tietueille. Kuitenkaan tässä sovelluksessa tälle toiminnallisuudelle ei ole tarvetta, vaikkei siitä ole sovelluksen toiminnan kannalta myös minkäänlaista haittaa.

Sovellus täyttää tuotteiden pudotusvalikon kaikilla tietokannasta saaduilla tuotteilla, sitoen tuotetunnukset listaesineiden nimikkeiksi ja ID:t arvoiksi, kuten nähdään kohdasta Ohjelma 7.

Lisäksi pudotusvalikon alkuun lisätään manuaalisesti listaesine oletusarvoiseksi valinnaksi. Lista esineen nimikkeeksi asetettiin ”valitse tuote”, joka on tarkoitettu kehotteeksi valita tuote, mutta myös implikoi, että tämän listaesineen ollessa valittuna sovellukseen ei ole valittu erillistä tuotetta.

Käyttäjän oletetaan ymmärtävän, että jos tuotetta ei ole erikseen valittu, sovellus näyttää kuvaajaa muodostettaessa historiadataa vain muita rajoituksia, kuten valittua aikaa, käyttäen.

4.2.5 Töiden mukaan hakeminen

Analyysinäkymään haluttiin mahdollisuus valita, minkä töiden historiadataa kuvaajan muodostuksessa käytetään. Tähän tarkoitukseen sivulle lisättiin valintaruutulista, jonka valintaruudut edustavat valittavana olevia töitä.

Linjaston tietokannassa on hieman hämäävästi nimetty ”Tilaukset” taulukko, johon on tallennettu kaikille linjaston töille niiden ID:t, työtunnukset, päättymisajat sekä niitä vastaavat tuotteet. Samaan tapaan kuin tuotteiden kanssa, töiden ID:t ovat tietokannan luomia automaattilukuja.

Sovellus hakee tietokannasta kaikki valittua tuotetta vastaavat työt sekä sitoo valintaruutulistan listaesineiden nimikkeisiin töiden tunnukset ja arvoihin töiden ID:t.

Jos tuotetta ei ole valittuna, sovellus hakee valintaruutulistaan kaikki linjaston tietokannan ”Tilaukset” taulukosta löytyvät työt.

Kun sovellus lopulta hakee historiadataa kuvaajan muodostamiseen, voidaan töiden valintaruutulistaa iteroida läpi ja lisätä valittujen listaesineiden arvot parametreiksi kyselyyn.

Jos yhtäkään työtä valintaruutulistasta ei ole valittu, sovellus hakee historiadataa työstä riippumatta muita rajoituksia käyttäen. Käyttäjän oletetaan ymmärtävän tämä sen perusteella, miten töiden valitsemisen käyttäjäkokemus on suunniteltu. Tarkempaa tietoa eri osa-alueiden käyttäjäkokemuksen suunnittelusta on luvussa 4.11 Käyttäjäkokemus.

Koska yksittäiselle työlle voi kertyä pitkän käytön aikana hyvinkin paljon töitä, voi haluttujen töiden löytäminen valintaruutulistasta käydä turhan työlääksi. Tätä varten töidenhakuun on lisätty mahdollisuus rajata näytettäviä töitä halutun ajan mukaan.

Töiden hakuun on lisätty tekstikentät aikaisinta ja viimeisintä näytettävää työn päättymisaikaa varten. Lisäksi molempia tekstikenttiä kohden on lisätty ASP.NET:n valmiiksi tarjoama kalenterinäkymä. Kalentereista valittaessa joku päivämäärä, sovellus päivittää vastaavaan tekstikenttään valitun päivämäärän halutussa formaatissa.

Ohjelma 8:ssa nähdään kuinka kalenterielementti ”TyötPVM” valittu päivämäärä muunnetaan tekstiksi ja muokataan haluttuun formaattiin sekä sijoitetaan tekstikentän ”TyöAika” tekstiksi.

Kalenterielementeissä valittu päivämäärä ei suoraan vaikuta töitä hakevaan kyselyyn, vaan nimenomaan tekstikentistä otetaan kyselyssä käytettävät parametrit. Tämä johtuu siitä, että pelkän kalenterielementin käyttö ei välttämättä tarjoa käyttäjän haluamaa tarkkuutta. Tekstikenttään käyttäjä voi lisätä halutessaan päivämäärän perään myös kellonajan.

Koska ASP.NET:n palvelinpuolen koodi käyttää C# ohjelmointikieltä, voidaan tekstikentillä syötettyjä tekstejä tulkita päivämääriksi ohjelmointikielen tarjoamalla ”TryParse” metodilla.

Tämä helpottaa käyttäjän toimintaa, sillä ”TryParse” metodi osaa ennakoida monia erilaisia formaatteja, jolloin annetun päivämäärän ja kellonajan formaatti ei ole niin jäykkä, kuin muissa sovelluksissa voisi olla.

Esimerkiksi ”1.1.1999” ja ”1. tammikuu 1999” tulkitaan samaksi päivämääräksi. Lisäksi kellonaikaa varten ei odoteta kaikki yksiköitä, vaan käyttäjä voi valita haluaako hakea töitä esimerkiksi minuuttien vai sekunnin murto-osien tarkkuudella.

Jos käyttäjä kuitenkin syöttää tekstikenttään täysin virheellisen ajan, sovellus ilmoittaa ponnahdusikkunalla käyttäjävirheestä.

Jos käyttäjä jättää esimerkiksi viimeisimmän näytettävän työn päättymisajan tekstikentän tyhjäksi, ei sovellus aseta takarajaa näytettäviä töitä hakevaan kyselyyn. Sama toimii myös aikaisimman näytettävän työn tekstikentän kanssa.

Jos molemmat kentät on jätetty tyhjiksi, sovellus ei rajaa näytettäviä töitä ajan perusteella. Käyttäjän oletetaan ymmärtävän tämä, sillä tekstikentät ovat oletuksena tyhjiä ja sovelluksessa on aikaisemminkin osoitettu ehtojen tyhjäksi jättämisen implikoivan ehdon käyttämättä jättämistä haussa.

Ohjelma 9:ssa on tarkemmin esitetty, miten sovellus lopulta hakee näytettävät työt ja sitoo datan ”Työt” valintaruutulistaan.

Koska töiden valinta ei ole suoraan analyysinäkymän sivun koodissa, vaan se on erotettu erilliseksi User Controlliksi, käytettään aiemmista ohjelmapätkistä poiketen sivun elementtien kutsumiseen ”FindControl” metodia. Tämä metodia hakee sivulta elementin, jolla on haluttu ID.

Töiden haun erottaminen User Controlliksi on tarkemmin selitetty luvussa 4.11 Käyttäjäkokemus.

4.2.6 Ajan mukaan hakeminen

Analyysinäkymään haluttiin mahdollisuus rajata kuvaajan muodostamiseen haettava historiadata ajan mukaan. Samaan tapaan kuin näytettäviä töitä rajattaessa ajan mukaan, historiadatan rajaamista varten on lisätty tekstikentät ja kalenterielementit edustamaan viimeisintä ja aikaisinta haettavaa dataa.

Samalla tavalla kuin töiden haussa, historiadatan ajan mukaan rajaamisen kalenterielementit päivittävät valitut päivämäärät tekstikenttiin, joihin käyttäjä voi tarkentaa tarvittaessa kellonajat ja joista sovellus saa parametrit kuvaajan muodostamiseen käytettävän historiadatan hakevaan kyselyyn.

Jos käyttäjä syöttää tekstikenttään virheellisen ajan, sovellus ilmoittaa ponnahdusikkunalla käyttäjävirheestä.

Koska käyttäjä oletettavasti haluaa historiadataa ajan mukaan rajaamalla rajoittaa töitä haettaessa asetettuja aikarajoja, päivittää sovellus automaattisesti töiden haun kalenterielementtien tai tekstikenttien arvon muuttuessa uudet arvot historiadatan ajan mukaan haun vastaaviin elementteihin. Tämä toiminnallisuus on tarkemmin selitetty luvussa 4.11 Käyttäjäkokemus.

4.2.7 Linjaston valinta

Koska eri linjastoilla on omat tuotteensa, työnsä ja historiadatansa, tarvitaan analyysinäkymään mahdollisuus valita mistä linjastosta käyttäjä haluaa hakea dataa.

Tähän tarkoitukseen sivulle on lisätty pudotusvalikko, jonka nimikkeiksi asetetaan linjastot numeroidusti ja arvoiksi linjastoja vastaavat Connection Stringit, kuten nähdään kohdasta Ohjelma 10.

Connection Stringit ovat tekstejä, joita vastaavat tarkemmat tietokantaan yhdistämiseen liittyvät tiedot on määritelty sovelluksen Web.config tiedostossa.

Esimerkiksi käytettäessä arvoa ”Linja1” metodissa, joka ottaa argumentiksi Connection Stringin, metodia saa Web.config tiedostosta tarkemmat tiedot miten otetaan yhteys Linjasto 1:n tietokantaan.

Tarkempaa tietoa Web.config tiedoston toiminnasta ja määrittelystä on luvussa 4.10.1 Web.config.

Koska sovelluksen nykyisessä käyttökohteessa tiedetään olevan vain kaksi erillistä linjastoa, voidaan linjastot ja niiden Connection Stringit lisätä staattisesti pudotusvalikon listaesineiksi sivun .aspx tiedoston HTML koodin.

Pudotusvalikon listaesineet voitaisiin myös määritellä ohjelmallisesti sivun palvelinpuolen koodissa, joka olisi tarpeellista, jos esimerkiksi haluttujen listaesineiden määrä ei ole aina sama. Kuitenkaan Connection Stringien ohjelmallinen määrittäminen ei ole suositeltavaa, eli linjastot lisätään todennäköisesti aina staattisesti.

4.2.8 Kuvaajan X-akselin vaihtaminen

Sovelluksen muodostamien kuvaajien oletusarvoinen x-akseli on aika-akseli, jossa datapisteiden x-arvona käytetään historiadataan tallennettua mittauksen aikaleimaa.

Sovellukseen haluttiin myös mahdollisuus vaihtaa jokin haluttu mittaus analyysinäkymän kuvaajan x-akseliksi.

Tähän tarkoitukseen analyysinäkymän sivulle lisättiin valintaruutu nimikkeellä ”Korrelaatio” sekä pudotusvalikko jonka eteen on asetettu nimike ”X-akseli”.

Pudotusvalikko on sivun ensilatauksessa oletusarvoisesti poistettu käytöstä, ja jos käyttäjä valitsee ”Korrelaatio” valintaruudun, pudotusvalikko otetaan käyttöön.

Käyttäjän oletetaan ymmärtävän, että ”Korrelaatio” valintaruudun valitsemalla hän haluaa vaihtaa kuvaajan x-akseli, eli tarkastella jonkun mittauksen korrelaatiota muihin mittauksiin.

”X-akseli” pudotusvalikkoon sidotaan samat mittaukset kuin mittauksien valintaan käytettyyn valintaruutulistaan, eli aina kun näytettäviä mittauksia rajoitetaan joilla aikaisemmin kuvatuilla käyttöliittymä toiminnallisuuksilla, vaikuttaa se samalla tavalla myös pudotusvalikon listaesineisiin.

4.2.9 Manuaalisesti lisättyjen pisteiden valinta

Sovelluksiin haluttiin mahdollisuus lisätä manuaalisesti mittauksia, joille annetaan lisättäessä aikaleima, työ- ja tuotetunnus, arvo sekä lisääjän nimi. Sovellukseen haluttiin myös mahdollisuus piirtää nämä pisteet analyysinäkymän kuvaajaan.

Tähän tarkoitukseen analyysinäkymän sivulle lisättiin valintaruutu nimikkeellä ”Manuaalinen data”.

Jos käyttäjä on valinnut valintaruudun, kuvaajaan haettaessa dataa haetaan myös valittuja hakuehtoja vastaavat manuaalisesti lisätyt datapisteet.

Koska manuaalisesti lisätyt datapisteet voidaan esimerkiksi lisätä linjaston ollessa pois käytössä ja pisteiden aikaleimat tai lisäystaajuus eivät muutenkaan luonnollisesti ole samat kuin historiadatan vastaavat, manuaalisesti lisättyjä pisteitä ja korrelaatiota ei voida käyttää samanaikaisesti.

Tätä varten sovellus asettaa automaattisesti korrelaation tai manuaalisesti lisättyjen pisteiden valinta ruudun pois käytöstä, jos toinen valintaruutu on jo valittuna.

Manuaalisesta datapisteiden lisäyksestä on tarkempaa tietoa luvussa 4.8 Manuaalinen datapisteiden lisäys.

4.2.10 Kuvaajan piirtäminen

Yllä olevissa kohdissa on selitetty kaikki käyttöliittymän toiminnallisuus, jonka käyttäjä tarvitsee muodostaakseen haluamanlaisensa kuvaajan.

Kun käyttäjä on tehnyt haluamansa valinnat käyttöliittymässä, kuvaaja muodostetaan painamalla painiketta, jonka nimike on ”Hae”. Käyttäjän oletetaan ymmärtävän hakemisella tarkoitettavan valitun datan hakemista ja siitä kuvaajan muodostamista.

Kun painiketta painetaan, sovellus tarkistaa onko valintaruutuvalikosta valittu vähintään yksi mittaus. Jos yhtäkään mittausta ei ole valittu, sovellus ilmoittaa käyttäjävirheestä ponnahdusikkunalla.

Data Access Objektin käyttö

Analyysinäkymän palvelinpuolen koodissa on määritelty instanssi ”PlotData” luokasta. Tämä objekti on analyysinäkymän Data Access Objekti (DAO), eli objekti jonka ominaisuuksiin on sijoitettu piirrettävät datat, sekä muita datan piirtoon liittyviä ominaisuuksia, kuten onko käyttöön valittu korrelaatio tai manuaalisesti lisätyt pisteet. Lisäksi objektilla on metodeja datan käsittelyyn kuvaajan piirtämistä varten.

Aluksi prosessi lisää DAO:iin listoihin valittujen mittauksien nimikkeet ja ID:t, kuten on esitetty kohdassa Ohjelma 11.

Yllä esitetyssä koodissa ”data” on viittaus analyysinäkymän DAO:iin.

Sovellus iteroi läpi valintaruutulistan ja lisää valittujen listaesineiden nimikkeet ja ID:t DAO:n vastaaviin listoihin.

Jos korrelaatio on valittu käyttöön, sovellus lisää korrelaatioon x-akseliksi valitun mittauksen listojen ensimmäiseksi esineeksi.

Koska kuvaajan nimikkeet näytetään kuvaajan selitteessä, kuvaajassa on käyräkohtaiset nimikkeet, eikä akselikohtaisia nimikkeitä. Tästä syystä valitun x-akselin nimikkeeksi lisätään tyhjä teksti.

Korrelaation ollessa valittuna, sovellus tarkastaa myös jokaisen valitun valintaruudun kohdalla, ettei valittu mittaus ole sama kuin valittu x-akseli.

Tällä tavalla käyttäjän ei tarvitse huolehtia sovelluksen piirtävän turhia kuvaajia, mihin on piirretty mittauksen korrelaatio itsensä kanssa.

Jos esimerkiksi käyttäjä valitsee valintaruutulistasta kolme mittausta ja haluaa tutkia niistä kahden korrelaatioita kolmanteen verrattuna, voi käyttäjä uutta kuvaajaa muodostettaessa vain vaihtaa haluttua x-akselia.

Kun haluttujen mittauksien nimikkeet ja ID:t on vastaavissa listoissaan, ohjelma hakee listojen perusteella linjaston tietokannan ”IO_DEF” taulusta mittauksien hakemiseksi tarvittavat tiedot, kuten on esitetty kohdassa Ohjelma 12.

”IO_DEF” taulussa on määritelty jokaiselle mittaukselle halutun historiadatan löytämiseen tarvitut arvot ”Block”, ”BlockNum” ja ”Block_Table”. Lisäksi taulusta haetaan ID:t palautetun datataulun järjestämistä varten.

Mittauksen ”Block_Table” viittaa, mistä tietokannan taulusta haluttu mittaus löytyy. Jos esimerkiksi ”Block_Table” kentän arvo on ”EX”, löytyy kyseisen mittauksen arvot taulukosta ”EX_PV_1”.

Jälkiliitteen ”PV” osa viittaa nykyisessä versiossa tauluihin, jotka sisältävät historiadataa ja jälkiviitteen ”1” osa on jäänne aikaisemmasta prototyypistä, jossa eri linjastojen taulukoiden ajateltiin olevan samassa tietokannassa.

Viimeisimmät mittaukset sisältävät taulut ovat erotettu jälkiliitteellä ”_MEAS_1”. ”IO_DEF” taulusta saatua ”Block_Table” arvoa voidaan siis käyttää taulun nimen etuliitteenä sekä viimeisimmän mittauksen että historiadatan hakemiseen.

Mittauksen ”BlockNum” on luku, joka vastaa tietokannan taulun saraketta, johon kyseisen mittauksen data tallennetaan. Mittaustauluilla on sarakkeet Data1:stä numerojärjestyksessä Data10:een asti. Jos esimerkiksi mittauksen ”BlockNum” kentän arvo on 3, tarkoittaa se, että kyseisen mittauksen arvot ovat vastaavan taulukon sarakkeessa ”Data3”.

Mittauksen ”Block” on luku, joka vastaa tietokannan taulun riviä, jolle on asetettu mittausta tallennettaessa sama arvo. Jos mittaus on ekstruuderikohtainen, ”Block” luku viittaa ekstruuderin numeroon, jolle mittaus kuuluu.

Projektin johdon antamien ohjeiden mukaan sovelluksella oletetaan olevan korkeintaan 10 ekstruuderia, ja ”Block” luvun ollessa yli 10 oletetaan sen vastaavan kyseistä lukua 10 pienempää arvoa. Esimerkiksi ”Block” luvun ollessa 11 sovellus käsittelee sitä samoin, kuin jos luku olisi 1. Tämä mahdollistaa useamman kuin 10 mittauksen kuulumisen yhdelle ekstruuderille, vaikka taulukossa onkin vain sarakkeet Data1-10. Koska nykyisessä käyttökohteessa tiedetään olevan 8 ekstruuderia per linjasto, voidaan sovellus huoletta toteuttaa tällä tavoin.

Kun mittauksien tiedot on haettu ”IO_DEF” taulusta, Data Access Layerin (DAL) palauttama datataulu järjestetään DAO:n listaan tallennettujen ID:ten mukaisesti ”AnalyysiDatat” luokan metodilla. Metodi on tarkemmin esitetty kohdassa Ohjelma 13.

Metodi käyttää Language Integrated Query (LINQ) kyselyä, joka valitsee datataulusta listan ID:tä vastaavan rivin ja lopuksi palauttaa järjestetyistä riveistä muodostetun datataulun. Jos järjestettävä taulu on tyhjä, metodi palauttaa niin ikään tyhjän taulun.

Mittauksien ”IO_DEF” taulusta haettujen tietojen järjestäminen ei ole analyysinäkymässä erityisen oleellista, kunhan korrelaation ollessa valittuna ensimmäinen mittaus on haluttu x-akseli.

Koska muissa sovelluksen ominaisuuksissa on tärkeämpää, että kyselyn palauttama taulukko on halutunlaisessa järjestyksessä, ja koska käyttäjäkokemuksen kannalta on miellyttävämpää, että kuvaajassa käyrät ovat samassa järjestyksessä kuin valitut mittaukset, on analyysinäkymässä käytetty samaa taulukon järjestämisen metodia kuin sovelluksen muissa osissa.

Kohdassa Ohjelma 14 esitetyssä koodissa ohjelma hakee DAO:n listaan valittuja ehtoja vastaavat aikaleimat. Taulukosta saadut aikaleimat ovat DateTime muodossa, mutta kuvaajan piirtoa varten ne muutetaan tekstiksi tiettyyn aikaformaattiin.

Toisin kuin aikaisemmissa kyselyissä, aikaleimaa haettaessa taulu on nimetty. Tämä johtuu siitä, että ”CmdEhto” luokan metodit, joilla kyselylle lisätään yllä mainituilla käyttöliittymän toiminnallisuuksilla tarkentavia ehtoja. ”CmdEhto” luokan toiminta on tarkemmin selitetty luvussa 4.4 Data Access Layer.

”AnalyysiDatat” luokan metodilla ”TäytäSuperlista” sovellus hakee DAO:n niin sanottuun superlistaan kuvaajaan halutut mittausdatat. Superlistoilla tarkoitetaan tämän sovelluksen yhteydessä listaa, joka käsittää muita listoja, joista jokainen käsittää yhden mittauksen haetut mittausarvot.

Metodi siis lisää DAO:n superlistan listoihin halutut mittausdatat. Listojen järjestys vastaa järjestetyn ID taulun järjestystä, sillä superlista täytetään iteroimalla kyseistä taulua.

Jos jollakin datapisteellä on tietokannassa tallennettu virheellinen tai puuttuva arvo, sovellus antaa pisteelle saman arvon kuin edelliselle arvolle.

Sovellus luo myös tyhjän listan structeille, jota superlistan täyttämisen metodi käyttää useaan kertaan saman mittausdatan turhaan hakemisen välttämiseen. Analyysinäkymässä toiminnolle ei ole tarvetta, mutta sovelluksen muissa osissa saatetaan tarvita samaa dataa useampaan tarpeeseen. Metodin tarkempi toiminta on kuvattu luvussa 4.4 Data Access Layer.

Kun sovellus on lisännyt DAO:iin mittauksien datat, sovellus lisää vielä haluttaessa manuaalisesti lisätyt datapisteet, kuten kohdassa Ohjelma 15.

Samaan tapaan kuin aikaleimoja lisättäessä, manuaalisesti lisättyjen datapisteiden lisäyksen kyselyssä taulu on nimetty. Manuaalista dataa varten DAO:lla on täysin muista mittauksista erilliset listat, koska manuaalisesti dataa lisättäessä aikaleimat ja lisäystaajuus eivät luonnollisesti ole samat kuin historiadatan vastaavat.

Lopuksi sovellus kutsuu DAO:n metodia ”ConvertList”, joka on esitetty kohdassa Ohjelma 16.

Metodi käyttää ”JavaScriptSerializer” luokan metodia, joka muuttaa C# listan JavaScriptin ymmärtämäksi JavaScript Object Notation (JSON) objektiksi. DAO:n listat myös tyhjennetään manuaalisesti, jotta objekti on kuvaajan piirron jälkeen palautettu alkuasemaansa.

Riippuen siitä onko korrelaatio tai manuaalisesti valitut datat otettu käyttöön, DAO muuttaa asianmukaiset listat JSON-objekteiksi, joita käyttäjäpuolen koodi lopulta lukee.

Listoja käännettäessä tiedetään jo, mikä datajoukko haluttiin kuvaajan x-akseliksi. Aikaleimalistan listaesineet ovat datatyypiltään tekstiä ja mittaukset ovat tyypiltään liukulukuja.

Koska listan JSON-objektiksi kääntäminen muuttaa listan joka tapauksessa tekstimuotoiseksi dataksi, ohjelman ei tarvitse ottaa huomioon datan tyyppiä x-akselia valittaessa.

Kun käyttäjä on painanut ”Hae” painiketta, ja yllä kuvattu prosessi on suoritettu palvelinpuolen koodissa, sivu laukaisee niin sanotun Post Back tapahtuman, jolloin sivu ladataan uudelleen.

Koska kuvaajan piirtäminen vaatii koko sivun uudelleen lataamista, on ”Hae” painike sijoitettu muista käyttöliittymän elementeistä poiketen Update Panelin ulkopuolelle. Jos painike olisi Update Panelin sisällä, Post Back päivittäisi vain Update Panelin sisäiset elementit.

Käyttäjäpuolen koodi kuvaajan muodostamisessa

Sivun uudelleen latauksessa Plotly kuvaajakirjastoa hyödyntävä JavaScript koodi saa DAO:n ominaisuuksista tarvittavat tiedot kuvaajan muodostamiseen.

Plotlyä käyttäville kuvaajille luodaan div-elementti sivun HTML koodiin. JavaScript koodissa kyseistä elementtiä kutsutaan ID:llä ja kirjasto piirtää elementtiin halutunlaisen kuvaajan.

Kohdan Ohjelma 17 koodissa kuvaajan div-elementille annetaan tyyli, jossa määrätään elementin korkeus ja leveys prosentteina siitä elementistä, jonka sisällä kuvaajan elementti on.

Koska sivun yläreunassa on jatkuvasti navigointipalkki ja sivun pohjalla on alaviite, kuvaajan elementin korkeudeksi on asetettu vain 90%.

Elementille lisätään myös Plotlyn tarjoama funktio, joka muuttaa kuvaajan elementin kokoa automaattisesti, jos selainikkunan kokoa muutetaan.

Kohdassa Ohjelma 18 on varsinainen koodi, jossa käyttäjäpuolen koodiin haetaan palvelinpuolelta DAO:sta tarvittava tieto ja kuvaaja muodostetaan.

Koodissa <% %> merkkien välinen koodi viittaa palvelinpuolen eli C# koodiin. Sovellus siis tarkastaa DAO:n ominaisuuksista onko manuaalisesti lisätyt datat tai korrelaatio valittu käyttöön.

Sovellus lisää palvelinpuolen koodissa määritellyn JSON-objektin kuvaajan x-akseliksi. Koska listoja sisältävä lista säilyttää rakenteensa, kun se käännetään JSON-objektiksi, voidaan sitä iteroida läpi käyttäjäpuolen koodissa ja lisätä sen sisältämät listat eri kuvaajan käyrien y-akseleiksi.

Koska nimikkeet sisältävä lista on samassa järjestyksessä kuin vastaavien mittauksien listat ovat DAO:n superlistassa, voidaan superlistan JSON muotoon käännettyä objektia iteroitaessa samalla indeksillä kutsua mittaukselle kuuluva nimike nimikkeiden JSON-objektista.

Datapisteiden joukkoa lisättäessä saadaan siis käännetyistä JSON-objekteista haluttu x- ja y-akseli sekä käyrän nimike. Käyrälle pitää vielä lisätä moodi ja piirtotyyppi.

Moodi tarkoittaa esimerkiksi minkälaisia merkkejä datapisteille käytetään ja onko pisteet yhdistetty jonkinlaisilla viivoilla. Käytetty ”markers” moodi tarkoittaa, että datapisteillä on yksinkertaiset pyöreät merkit, eikä datapisteitä ole yhdistetty toisiinsa.

Piirtotyyppi määrää, minkälainen kuvaaja piirretään ja käytetäänkö piirtämiseen WebGL:ää vai SVG:tä. Valittutyyppi ”scattergl” tarkoittaa yksinkertaista kuvaajaa, jossa datapisteet ovat xy-tasossa ja piirtämiseen käytetään WebGL. WebGl on valittu käyttöön, koska se soveltuu paremmin suurien pistemäärien piirtämiseen.

Kun käyrän ”Trace” datat ja ominaisuudet on asetettu, lisätään sen array ”Traces” komennolla ”Traces.push(Trace)”.

Aikaisemmin selitetyistä syistä manuaalisesti lisätyille pisteille on täysin erilliset x- ja y-akselit, eli niiden piirtäminen on erillään myös käyttäjäpuolen koodissa.

Kuvaajalle lisätään myös objekti, jossa määritellään kuvaajan yleinen ulkoasu. Tässä tapauksessa ulkoasuun haluttiin vain lisätä selite, josta nähdään käyrien nimikkeet.

Lopulta varsinainen kuvaaja luodaan Plotlyn funktiolla, jolle annetaan haluttu div-elementti, johon kuvaaja piirretään, array, joka sisältää kuvaajaan halutut käyrät ja ulkoasuasetukset sisältävä objekti.

4.2.11 Uusien vertailuarvojen laskenta

Analyysinäkymään haluttiin toiminnallisuus laskea valitusta datasta vertailuarvot, joita voidaan käyttää reaaliaikanäkymässä viimeisimpien mittaustulosten tarkasteluun. Tätä varten analyysinäkymässä on painike nimikkeellä ”Uudet vertailuarvot”.

Painiketta painettaessa sovellus laskee valitulta ajalta, valituista töistä ja tuotteesta vertailuarvot sovelluksen yhteiseen tietokantaan tallennettujen laskuohjeiden mukaan. Käyttäjän oletetaan ymmärtävän käyttöliittymä toiminnallisuuksien, joilla rajataan kuvaajaan haettavaa dataa vaikuttavan myös vertailuarvojen laskentaan. Vertailuarvojen vertailu on tarkemmin selitetty luvussa 4.5 Vertailuarvot.

Jos käyttäjä ei ole valinnut tuotetta, sovellus ilmoittaa ponnahdusikkunalla käyttäjävirheestä, sillä vertailuarvot ovat aina sidottuja tiettyyn tuotetunnukseen.

Ennen kuin vertailuarvoja aletaan laskea, käyttäjältä varmistetaan, että hän haluaa laskea uudet vertailuarvot, sillä laskenta edellyttää vanhojen arvojen korvaamista. Varmistus tapahtuu ponnahdusikkunalla, jossa käyttäjällä on mahdollisuus varmistaa laskennan aloitus tai peruuttaa laskenta.

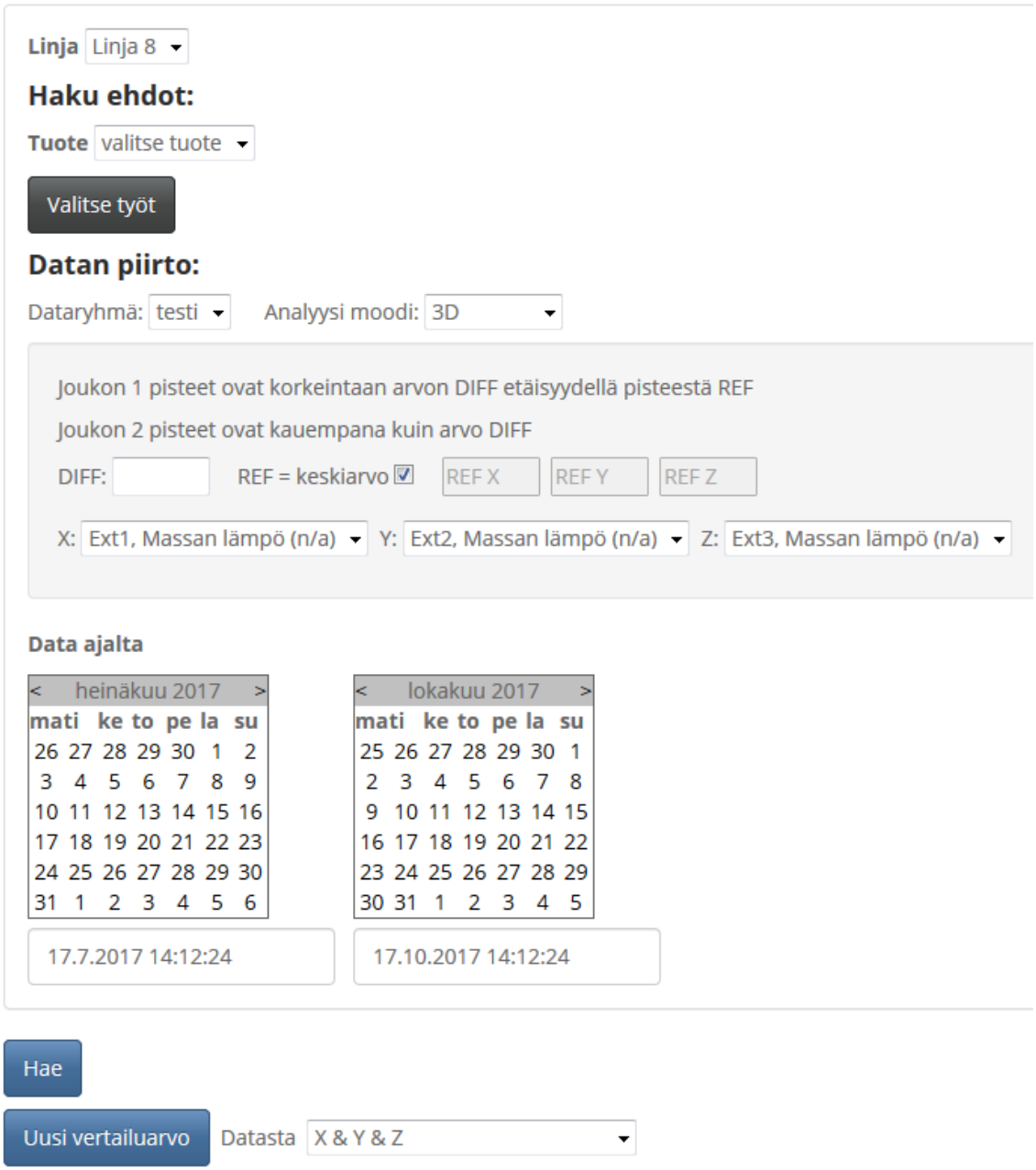
Kun vertailuarvot on laskettu ja tallennettu onnistuneesti, sovellus ilmoittaa käyttäjälle ponnahdusikkunalla, jossa on esitetty lasketut linjamittauksien keskiarvot. Vain lasketut keskiarvot esitetään, koska kaikkien vertailuun laskettujen arvon esittäminen veisi liikaa tilaa.

4.3 Luokka-analyysinäkymä

Sovellukseen haluttiin mahdollisuus muodostaa 3D-kuvaajia, joilla voitaisiin tutkia kolmen eri mittauksen keskinäisiä korrelaatioita.

Tähän tarkoitukseen sovellukseen lisättiin niin sanottu luokka-analyysinäkymä, joka suurimmilta osin vastaa analyysinäkymää, mutta luokka-analyysin kuvaajat ovat aina 3D-kuvaajia joiden muodostamiseen käytetään kolmea tai neljää mittausta riippuen valitusta moodista.

Tiettyjen kuvaajan muodostamiseen liittyvien toiminnallisuuksien vuoksi analyysinäkymää ei ole yhdistetty luokka-analyysinäkymään yhdeksi moodiksi.



Kuva 7 Luokka-analyysinäkymä

Kuvassa Kuva 7 on esitetty lopullinen versio luokka-analyysinäkymästä, jonka eri toiminnallisuudet on selitetty tämän kappaleen alaosioissa.

4.3.1 Analyysinäkymän kanssa yhteiset toiminnallisuudet

Koska sekä analyysi- että luokka-analyysinäkymässä päätoiminnallisuus on kuvaajan muodostaminen tai vertailuarvojen laskeminen, on molemmissa sivuissa luonnollisesti paljon samoja haettavan datan rajaamiseen liittyviä käyttöliittymän toiminnallisuuksia.

Linjaston ja tuotteen valinnan pudotusvalikot ovat toteutukseltaan molemmilla sivuilla identtiset. Myös datan ajan mukaan rajaamiseen käytetyt elementit ovat molemmissa sivuissa samat.

Jos molemmissa sivuissa kutsutaan täysin samaa koodia, on koodi järkevämpi erottaa omaan luokkaansa sen sijaan, että molempien sivujen code behind tiedostossa on sama koodi. ”YhteisetEventit” luokkaan on koottu metodeja, joita molemmat kuvat sivut kutsuvat tiettyihin elementteihin sidottujen tapahtumien yhteydessä.

Esimerkiksi ajan rajaamisen kalenterista valittaessa uusi päivämäärä, sivu kutsuu metodia erillisestä luokasta, sillä molemmissa sivuissa kalenterin halutaan päivittävän vastaavan tekstikentän sisältö samalla tavalla.

Töiden valinta on erotettu omaksi User Controlliksi, mikä tarkoittaa sitä, että sama HTML koodi voidaan lisätä muutamalla rivillä molemmille sivuille. Lisäksi User Controlleilla on omat erilliset code behind tiedostonsa, eli töiden mukaan hakemisen tapahtumien koodia tai muuta koodia ei tarvitse erikseen erottaa omaan luokkaansa.

Kohdasta Ohjelma 19 nähdään, miten sivun HTML koodissa käsitellään User Controllia. Ensimmäisessä rivissä sivulle rekisteröidään haluttu User Control, joka on tässä tapauksessa töiden valinnan ponnahdusikkuna. Toisella rivillä sivulle lisätään rekisteröinnissä määritelty elementti.

Näytettäviksi mittauksiksi luokka-analyysinäkymään haluttiin vain mittausryhmät, eli erillistä elementtiä näytettävien mittauksien seulontaa ei tarvita.

Mittausryhmien pudotusvalikkoon ja mittauksien valinta elementteihin datan sidonta toimii samalla tavalla kuin analyysinäkymässä, mutta toisin kuin analyysinäkymässä, luokka-analyysinäkymässä mittaukset valitaan neljästä pudotusvalikosta.

Koska luokka-analyysi kuvaaja voidaan piirtää vain kolmea tai neljää mittausta käyttäen, voidaan käyttää staattista määrää käyttöliittymä elementtejä, joista käyttäjä voi valita suoraan tietyn mittauksen tietylle akselille.

4.3.2 3D-moodi

Luokka-analyysin oletusmoodi tai perusmoodi on niin kutsuttu 3D-moodi, jossa yksinkertaisesti valitaan kolme mittausta 3D-kuvaajan x-, y- ja z-akseleiksi.

Sovellus myös laskee akseleiden keskiarvot ja muodostaa niistä vertailupisteen. Käyttäjä valitsee haluamansa etäisyyden, ja tämän etäisyyden mukaan kuvaajaan piirrettävät pisteet jaetaan kahteen joukkoon: pisteet jotka ovat asetettua etäisyyttä kauempana vertailupisteestä ja pisteet jotka ovat korkeintaan etäisyyden päässä vertailupisteestä.

Käyttäjä voi myös halutessaan antaa oman vertailupisteen. Tätä varten sivulle on valinta ruutu nimikkeellä ”keskiarvo”, joka on oletusarvoisesti valittuna. Jos käyttäjä poistaa valinnan, sovellus aktivoi kolme tekstikenttään, joihin käyttäjä voi syöttää haluamansa pisteen x-, y- ja z-koordinaatit.

Jos käyttäjä antaa virheellisen etäisyyden tai koordinaatin, sovellus ilmoittaa käyttäjävirheestä ponnahdusikkunalla.

4.3.3 Ulkorajamoodi

Luokka-analyysin ulkorajamoodissa käyttäjä valitsee kolme mittausta 3D-kuvaajan kolmeksi akseliksi ja lisäksi neljännen mittauksen. Käyttäjä määrittelee neljännelle mittaukselle ala- ja ylärajat. Kun kuvaaja muodostetaan, sovellus jakaa pisteet joukkoon: pisteet joilla neljännen mittauksen arvo on alle alarajan ja pisteet joilla mittaus on yli ylärajan.

Jos käyttäjä haluaa jakaa datapisteet kahteen joukkoon jonkin raja-arvon mukaan, voi käyttäjä asettaa molemmat rajat samaksi.

Koska rajat käytännössä katsoen rajaavat pois osan datapisteistä, täytyy ulkorajamoodissa käyttäjän antaa molemmille rajoille kelvolliset arvot. Jos rajoille on annettu virheellinen arvo tai arvoa ei ole annettu, sovellus ilmoittaa käyttäjävirheestä ponnahdusikkunalla.

Kuten edellä mainittiin, kaikille mittauksille, mukaan lukien vaihtoehtoiselle neljännelle, on oma staattinen pudotus valikko. Kuitenkaan neljännen mittauksen pudotusvalikkoa ei tarvita 3D-moodissa ja vastaavasti 3D-moodin etäisyyden syöttöön käytettyä teksti kenttää ei tarvita ulkorajamoodissa.

Tätä varten sovellus piilottaa ja poistaa käytöstä käyttöliittymä elementtejä moodia vaihdettaessa sen mukaan mikä moodi valitaan.

4.3.4 Sisärajamoodi

Samaan tapaan kuin ulkorajamoodissa, sisärajamoodissa käyttäjä valitsee neljännen mittauksen ja sille ylä- ja alarajan. Sisärajamoodissa sovellus yksinkertaisesti hakee datapisteet, joiden neljännen mittauksen arvo on valittujen rajojen välissä.

Muuten sisärajamoodi toimii samalla tavalla kuin 3D-moodi, eli sovellus laskee valituista kolmesta akselista keskiarvopisteen tai käyttäjä antaa oman vertailupisteensä ja käyttäjä antaa etäisyyden, jonka mukaan datapisteet jaetaan joukkoihin.

Jos jommallekummalle rajalle on annettu virheellinen arvo tai jos arvoa ei ole annettu, sovellus ilmoittaa ponnahdusikkunalla käyttäjävirheestä.

4.3.5 Väriskaalamoodi

Väriskaalamoodissa käyttäjä valitsee neljännen mittauksen. Kuvaajaa muodostettaessa datapisteille annetaan väri neljännen mittauksen mukaan, eli neljättä mittausta käytetään niin sanottuna väriskaalana.

Tämä mahdollistaa jonkun asteisen neliulotteisen kuvaajan luomisen. Koska datapisteiden eri joukkoihin jakaminen visualisoidaan pisteiden värillä, ei väriskaalamoodissa käytetä akselien keskiarvon laskemista, vaikka moodi muuten muistuttaakin 3D-moodia.

4.3.6 Kuvaajan muodostaminen

Koska analyysinäkymästä poiketen luokka-analyysinäkymässä datapisteille on useampia piirrettäviä joukkoja, luokka-analyysiä varten on luotu oma erillinen DAO luokka.

Kuvaajan muodostaminen tapahtuu suurimmilta osin samalla tavalla kuin analyysinäkymässä. Mittauksien valinnan pudotusvalikoista saadaan haluttujen mittauksien ID:t joiden perusteella haetaan tarkemmat tarvittavat tiedot ”IO_DEF” taulusta. DAO:lla on superlista, johon lisätään listoina valittujen akselien mittausdatapisteet.

Toisin kuin analyysinäkymässä, luokka-analyysissä sovellus joutuu iteroimaan superlistansa läpi mahdollisesti useampaan kertaan. Ensin sovellus iteroi superlistaa laskeakseen x-, y- ja z-akselien arvoista keskiarvopisteen, jos keskiarvopistettä käytetään valitussa moodissa, kuten kohdassa Ohjelma 20. Seuraavaksi sovellus iteroi superlistan ja jakaa datapisteet joukkoihin käyttäjän moodista riippuen antamien arvojen mukaan.

Keskiarvoa laskettaessa käytetään C# kielen tarjoamaa niin sanottua ”Try Catch” rakennetta, joka automaattisesti havaitsee mahdollisia poikkeuksia ohjelman suorittamisessa. Keskiarvon laskemissa rakennetta on käytetty, koska jos kaikilla akseleilla ei ole samaa määrää pisteitä kuin x-akselilla, aiheutuu poikkeustilanne.

Kuten kohdasta Ohjelma 20 nähdään, sovellus yrittää hakea arvoa indeksillä kaikkien mittauksien listoista, ja jos jostain listasta puuttuu arvo, sovellus ei lisää pistettä keskiarvoon.

Lisäksi sovellus lisää pisteiden arvot teksti muodossa StringBuilder objekteihin, jota tarvitaan vertailuarvon laskemista varten. Jokaiselle kolmesta akselista on kolme StringBuilder objektia, joista yhteen kootaan kaikki akselin mittaukset ja muihin kahteen lisätään tarpeen mukaan pisteet joukkojen mukaan samalla, kun sovellus jakaa pisteitä joukkohin.

Kun StirngBuilderiin lisätään pisteen arvo, lisätään arvon perään välimerkki, jonka perusteella tekstimuotoinen lista voidaan myöhemmin jakaa uudestaan lista muotoon.

Kun halutut historiadatan arvot on haettu DAO:iin, samaan tapaan kuin analyysinäkymässä objekti kääntää datalistat JSON-objekteiksi käyttäjäpuolen koodia varten.

Luokka-analyysinäkymän käyttäjäpuolen koodissa käännetyt JSON-objektit lisätään datapistejoukkojen akseleiksi, mutta luonnollisesti toisin kuin analyysinäkymässä joukoilla on kolme akselia.

Luokka-analyysinäkymässä joukkoja ei luoda ohjelmallisesti vaan jos käytetyssä moodissa datapisteet on jaettu jonkin kriteerin mukaan, ovat kuvaajan joukot samat kuin jaossa määritellyt. Jos moodissa käytetään vertailupistettä, vertailupisteelle muodostetaan oma yhdenpisteen joukko. Jos taas käytössä on väriskaalamoodi, kuvaajaan piirretään vain yksi pistejoukko, jolle asetetaan neljäs mittaus väriskaalan skaalaksi, kuten nähdään kohdassa Ohjelma 21.

Kuvaajan muodostamisen yhteydessä sovellus lisää sivulla oleviin Hidden Field elementteihin StringBuildereihin kootut datapistelistat. Pisteiden arvot koottiin StringBuildereihin, koska Hidden Field elementtien arvot voivat olla vain tekstimuodossa.

Hidden Field elementit ovat käyttäjälle näkymättömiä HTML elementtejä, joihin voidaan tallentaa jokin tekstimuotoinen arvo. ASP.NET sivujen elinkaaresta johtuen sivun täyden uudelleenlataamisen yhteydessä tietoa joudutaan tallentamaan väliaikaisesti Hidden Field elementteihin, jos sitä halutaan käyttää uudelleenlatauksen jälkeen. Koska kuvaajat käyttävät Plotlyä, sivu täytyy ladata uudelleen kuvaajaa muodostettaessa. DAO:iin tallennetut tiedot eivät säily sovelluksen muistissa sivun uudelleenlatauksessa.

Datapistelistojen lisäksi Hidden Field elementteihin tallennetaan kuvaajaa piirrettäessä käytetyt Connection String, tuotetunnus ja mittauksien ID:t, joita tarvitaan vertailuarvon laskennassa.

4.3.7 Vertailuarvon laskenta

Kuten analyysinäkymässä, luokka-analyysinäkymässäkin on mahdollisuus laskea vertailuarvoja reaaliaikanäkymän vertailua varten.

Analyysinäkymästä poiketen, luokka-analyysissä vertailuarvon laskentaan käytettävä data pitää aina piirtää ensin kuvaajaan. Tämä johtuu siitä, että luokka-analyysinäkymässä käyttäjä voi valita lasketaanko uusi vertailuarvo kaikista datapisteistä vai jostain tietystä kuvaajassa erotellusta pistejoukosta.

Luokka-analyysinäkymässä voi myös tallentaa vain yhden vertailuarvon kerrallaan, kun taas analyysinäkymässä lasketaan aina kaikki tietokannassa määritellyt vertailuarvot.

Samaan tapaan kuin analyysinäkymässä, käyttäjältä varmistetaan ponnahdusikkunalla, haluaako hän varmasti laskea uuden vertailuarvon. Analyysinäkymästä poiketen luokka-analyysinäkymässä käyttäjä voi myös valita kuinka monesta mittauksesta vertailuarvo lasketaan.

Järjestelmän yksinkertaistamiseksi tarjotut vaihtoehdot on esitetty pudotusvalikossa, josta käyttäjä voi valita joko x-, x- ja y- tai x-, y- ja z-akselit käytettäväksi vertailuarvon laskemiseen. Lisäksi käyttäjä voi valita x-, y-, z- ja f-akselit, jossa f-akseli on neljäs mittaus, joka on joissain moodeissa käytössä.

Jos käyttäjä valitsee useamman mittauksen vertailuarvon laskentaan kuin tietokannan vertailuarvot sisältävän taulun ”RT_RefArvot” muotoilu tukee, sovellus ilmoittaa ponnahdusikkunalla käyttäjävirheestä.

Jos käyttäjä valitsee vertailuarvon laskemisen vain yhden mittauksen (akselin) käyttäjä voi halutessaan valita laskettavan vertailuarvon olevan niin sanotusti lineaarinen vertailu. Vertailuarvojen laskeminen on selitetty tarkemmin luvussa 4.5 Vertailuarvot.

Kun vertailuarvoa lasketaan, sovellus saa laskentaan käytetyt datapisteiden arvot valittua joukkoa vastaavilta Hidden Field elementeiltä. Sovellus pilkkoo tekstimuodossa olevan listan lisätyn välimerkin perusteella ja kääntää ne takaisin liukulukulistaksi.

Lisäksi sovellus tarkastaa Hidden Field elementtiin tallennetut mittauksien ID:t. Jos yhtäkään ID:tä ei ole tallennettu kuvaajan muodostamisen yhteydessä, sovellus ilmoittaa virheellisestä käytöstä ponnahdusikkunalla, sillä luokka-analyysinäkymässä vaaditaan aina kuvaajan muodostaminen ennen kuin vertailuarvoja voi laskea.

Sovellus myös huomauttaa käyttäjävirheestä, jos valituissa mittauksissa on kaksi tai useampi samaa mittausta.

Jos valitut mittaukset eivät vastaa mitään tallennettua vertailuarvoa, sovellus huomauttaa tästä käyttäjälle ponnahdusikkunalla ja varmistaa että käyttäjä haluaa jatkaa vertailuarvon tallentamista. Tämä johtuu siitä, että jos käyttäjä on jo laskenut kaikki vertailuarvot analyysinäkymässä, vertailuarvon pitäisi olla jo tallennettuna, jos käyttäjä haluaa, että sitä käytetään reaaliaikanäkymän vertailussa. Reaaliaikanäkymän tarkemmasta toiminnallisuudesta on kerrottu luvussa 4.6 Reaaliaikanäkymä.

Koska vertailuarvot ovat aina sidottu johonkin tuotetunnukseen, sivun Hidden Field elementtiin tallennettiin kuvaajan muodostamisen yhteydessä valittu tuotetunnus. Jos tuotetunnusta ei valittu ja käyttäjä yrittää laskea vertailuarvoa, sovellus ilmoittaa käyttäjävirheestä ponnahdusikkunalla.

Vertailuarvot ovat oletusarvoisesti linjastokohtaisia, joten sovellus tallentaa kuvaajan muodostamisen yhteydessä käytetyn Connection Stringin omaan Hidden Field elementtiinsä. Hidden Field elementtejä käyttämällä sovellus saa siis vertailuarvon tallentamiseen tarvitun Connection Stringin ja tuotetunnuksen.

Analyysinäkymästä poikkeavien toiminnallisuuksien tarkoitus on tarjota käyttäjälle mahdollisuus korjata yksittäisiä vertailuarvoja ja tähän tarkoitukseen käyttäjälle on tarjottu tarkempi mahdollisuus rajata laskentaan haluttu data.

4.4 Data Access Layer

Sovelluksen DAL:lla voidaan katsoa olevan kaksi tasoa. Niin sanotusti syvempi taso, joka on suoraan yhteydessä tietokantaan, on luokka, jonka metodeilla varsinaiset kyselyt lähetetään tietokantaan.

Syvempää tasoa käytetään kaikissa tapauksissa, joissa sovellus tarvitsee suoraan taulun tai arvon tietokannasta. Lisäksi DAL:n ylempi taso käyttää historiadatan hakuun alempaa tasoa.

Nimenomaan kuvaajien muodostamista varten dataa haettaessa käytetään luokan metodeja, joiden voidaan ajatella olevan DAL:n ylempi taso. Metodit käyttävät alemman tason metodeja historiadatan noutamiseen tietokannasta ja sijoittavat datat käytetyn sivun DAO:n superlistoihin.

4.4.1 Syvempi taso

DAL:n syvempi taso on käsitetty luokassa ”DByhteys” johon kuuluu kolme staattista metodia.

Ensimmäistä metodia käytetään hakemaan datataulu tietokannasta. Kyselyssä taulu määritellään kyselyn komennossa annettujen ehtojen mukaisesti. Ehtojen muuttujat on lisätty komentoon parametreinä. Metodi on esitelty kohdassa Ohjelma 22.

Metodissa luodaan OleDb data-adapteri käyttämällä annettua kyselyä ja Connection Stringiä. Data-adapterilla tietokannasta saadut tiedot sijoitetaan datataulun, ja lopuksi sovellus sulkee yhteyden ja palauttaa muodostetun taulun.

Tason toinen metodi on vastaava kuin yllä kuvattu metodi, mutta se on tarkoitettu kyselyille joissa ei tarvita parametrejä. Tästä syystä toiseen metodiin ei tarvita OleDb komento-objektia, vaan riittää että parametrinä annetaan kysely suoraan tekstimuodossa.

Kolmatta metodia käytetään tietokannan muokkaamiseen tarkoitettujen komentojen lähettämiseen. Metodi on esitetty kohdassa Ohjelma 23.

Samaan tapaan, kuin dataa tietokannasta noudettaessa, OleDb komennon kyselyssä on määritelty muokkaamisen ehdot ja ehtojen muuttujat on lisätty komentoon parametreinä.

Näitä metodeja kutsumalla sovelluksen muut osat ja DAL:n ylempi taso ovat yhteydessä tietokantoihin.

4.4.2 Ylempi taso

DAL:n ylempää tasoa käytetään kuvaaja muodostettaessa DAO:n superlistan täyttämiseen. Taso on käsitetty luokassa ”AnalyysiData”, jonka aiemmin mainittua ”TäytäSuperlista” metodia kutsutaan kuvaajaan muodostettaessa.

Metodille annetaan parametreiksi käyttäjän käyttöliittymän toiminnallisuuksilla määrittelemät ehdot, kuten haluttu tuotetunnus. Parametriksi annetaan myös taulukko, johon sovellus on kuvaajan muodostamisen aikaisemmassa vaiheessa hakenut jokaiselle halutulle mittaus ID:lle tarkemmat tiedot, jotka tarvitaan historiadatan noutamiseen.

Metodille annetaan myös totuusarvomuuttuja, joka määrää ollaanko dataa hakemassa historiadatataulukoista vai viimeisimmät mittaukset sisältävistä tauluista.

Metodille annetaan myös käytetty superlista sekä dataa hakevan prosessin alussa määritelty tyhjä struct-lista.

Struct-listaan lisätään uusi objekti jokaista uutta haettua mittausta varten, ja objektiin tallennetaan mittauksen ID sekä lista, johon on tallennettu mittaukselle haettu historiadata.

Kun sovellus hakee uusia mittauksia superlistaa täytettäessä, sovellus tarkastaa jokaisen ID:n kohdalla onko kyseisen ID:n historiadata jo haettu. Jos historiadata on jo haettu, sovellus voi lisätä struct-listasta suoraan ID:tä vastaavan historiadata listan superlistaan. Näin vältetään turhia samoja kyselyitä, jotka hidastavat prosessia.

Mittauksen datan hakeminen

DAL:n ylempi taso iteroi läpi mittauksien ID taulua ja muodostaa taulusta saatujen ”Block”, ”BlockNum” ja ”Block_Table” avulla kyselyn, jolla halutun ID:n historiadata löydetään tietokannasta.

Sovellus käyttää ”CmdEhto” luokan metodeja muodostaakseen varsinaisen komennon, jolla saadaan haluttu historiadata datatauluna DAL:n syvemmän tason metodia käyttämällä. Datataulusta saadaan arvot superlistan listaan iteroimalla taulun rivejä läpi, kuten nähdään kohdassa Ohjelma 24.

”CmdEhto” luokassa sovellus muodostaa varsinaisen komennon annettujen ehtojen avulla. Luokan metodi, jolla komentoon lisätään aikarajat sekä työ- ja tuotetunnuksien mukaiset hakuehdot, on esitelty kohdassa Ohjelma 25.

Avainarvon datan hakeminen

Sovelluksen iteroidessa ID taulua, jos ”Block_Table” arvona on ”KEY”, sovellus tietää, että kyseinen mittaus on niin sanottu avainarvomittaus, jonka arvot lasketaan useammasta mittauksesta. Sovellus siis päättää ID taulua iteroidessa kutsutaanko normaalia yksittäisen mittauksen hakemisen metodia vai erillistä avainarvojen hakemisen metodia.

Avainarvon hakemisessa on määritelty tässä käyttötapauksessa kaksi laskutapaa, joita sovellus käyttää linjaston tietokannan ”KEYCALC” taulusta saamiensa tietojen mukaan.

Taulussa on rivi joka vastaa avainarvomittauksen ID:tä. Rivillä on määritelty kumpaa laskumenetelmää arvoja laskettaessa käytetään, mitä mittauksia laskennassa käytetään sekä joitain haluttuja vakio arvoja, joita tarvitaan mittauksessa.

Käytännössä avainarvoa laskettaessa sovellus luo uuden superlistan, johon sovellus hakee ”TäytäSuperlista” metodilla mittausdatat. Käytännössä siis avainarvot voisivat sisältää toisia avainarvoja, mutta luonnollisesti mitä useampi mittaus lopulta joudutaan hakemaan, sitä pitempään prosessin suorittamisessa kestää.

4.5 Vertailuarvot

Sovellus käyttää vertailuarvoja reaaliaikanäkymässä haluttujen arvojen seurantaan. Vertailuarvoja voidaan laskea kaikki kerralla analyysinäkymässä tietokannassa määriteltyjen ohjeiden mukaan tai yksi kerrallaan luokka-analyysinäkymässä.

Vertailuarvot voidaan jakaa kahteen joukkoon: lineaariset vertailuarvot ja ekstruuderin vertailuarvot.

Ekstruuderin vertailuarvot on kuitenkin vain käytetty termi, ja lineaarinen vertailuarvo voi perustua niin yksittäisen ekstruuderin kuin koko linjastonkin mittaukseen.

Kuitenkaan ekstruuderin vertailuarvot eivät ole käytännössä koskaan täysin linjastolle yhteisiä, vaikka joku niiden laskentaa käytetty mittaus voikin olla linjastolle yhteinen. Pelkkiä linjaston mittauksia käyttävä ekstruuderin vertailuarvo sisältäisi saman vertailuarvon jokaista ekstruuderia kohden.

Tietokannan taulussa on määritelty vertailuarvoille laskusäännöt, joissa kerrotaan, kummanlainen vertailuarvo on kyseessä ja mitä mittauksia laskennassa käytetään.

Taulua iteroimalla sovellus hakee ekstruuderikohtaisille mittaukselle ensin kaikki yhden ekstruuderin mittauksia vastaavien ID:en arvot ja laskee niistä halutut arvot. Sovellus sitten jatkaa hakemalla seuraavan ekstruuderin ID:t ja niin edelleen, kunnes vertailuarvot on laskettu kaikille ekstruudereille. Jos vertailuarvo on linjakohtainen, sovellus laskee luonnollisesti vain yhden vertailuarvon.

Koska ”IO_DEF” taulukossa kaikille eri ekstruudereiden vastaaville mittauksille on asetettu sama nimi, voidaan vertailuarvojen laskusäännöt sisältävässä taulussa käyttää mittausten nimiä ja sovellus löytää tarvittavat ID:t ekstruuderikohtaisesti nimen perusteella.

Lineaariset vertailuarvot

Lineaarinen vertailuarvo koostuu yksinkertaisesti valitusta datasta lasketusta keskiarvosta ja annetusta niin sanotusta varmuuskertoimesta.

Reaaliaikanäkymässä sovellus hakee tietokannasta halutuille mittauksille lineaarisen vertailuarvon ja varmuuskertoimen ja jos viimeisin mitattu arvo eroaa keskiarvosta yli varmuuskertoimen verran, reaaliaikanäkymässä varoitetaan arvon poikkeamasta käyttöliittymäelementillä.

Ekstruuderin vertailuarvot

Ekstruuderin vertailuarvot muodostetaan yleensä useammasta mittauksesta, joille lasketaan keskiarvot ja hajonnat. Arvot lasketaan kaikille linjaston ekstruudereille, eli tietokannassa on yhtä vertailuarvoa kohden ekstruudereiden määrän mukainen määrä rivejä.

Koska riveihin tarvitaan jokaista yhtä käytettyä mittausta kohden erillinen sarake mittauksen ID:tä, keskiarvoa ja hajontaa varten, on ekstruuderin vertailuarvot sisältävän taulun muotoilu oleellista tehdä halutun maksimi mittausmäärän mukaiseksi.

Sovellus laskee vertailuarvolle ylärajan mittauksien hajontaan suhteutetun keskiarvon ja hajonnan määrän mukaan.

Jos reaaliaikanäkymässä sovelluksen laskema viimeisimpien mittauksien muodostaman piste on yli lasketun ylärajan, reaaliaikanäkymässä varoitetaan arvon poikkeamasta käyttöliittymä elementillä.

4.6 Reaaliaikanäkymä

Sovellukseen haluttiin mahdollisuus seurata linjastojen toimintaa reaaliajassa. Näkymästä haluttiin kaksi versiota: yksi näkymä, johon vain piirretään tietyt linjastojen mittaukset viimeisen kahden vuorokauden ajalta omiin linjastokohtaisiin kuvaajiinsa, sekä näkymä, jossa kuvaajien lisäksi on käyttöliittymäelementtejä, joista nähdään, poikkeaako mitattu arvo liikaa tallennetusta vertailuarvosta,

Kuvaajien piirtäminen toimii yksinkertaisesti ja lähes samalla tavalla kuin analyysinäkymässä.

Reaaliaikanäkymässä oletetaan haluttujen näytettävien mittauksien olevan aina samat, eli dataa hakevan kyselyn ehdot on voitu nykyisessä toteutuksessa määritellä suoraan koodissa.

Arvojen seurantaelementit sovellus generoi ohjelmallisesti. Sovellus käyttää generointiin samaa taulukkoa, josta vertailuarvojen laskenta saa laskusääntönsä.

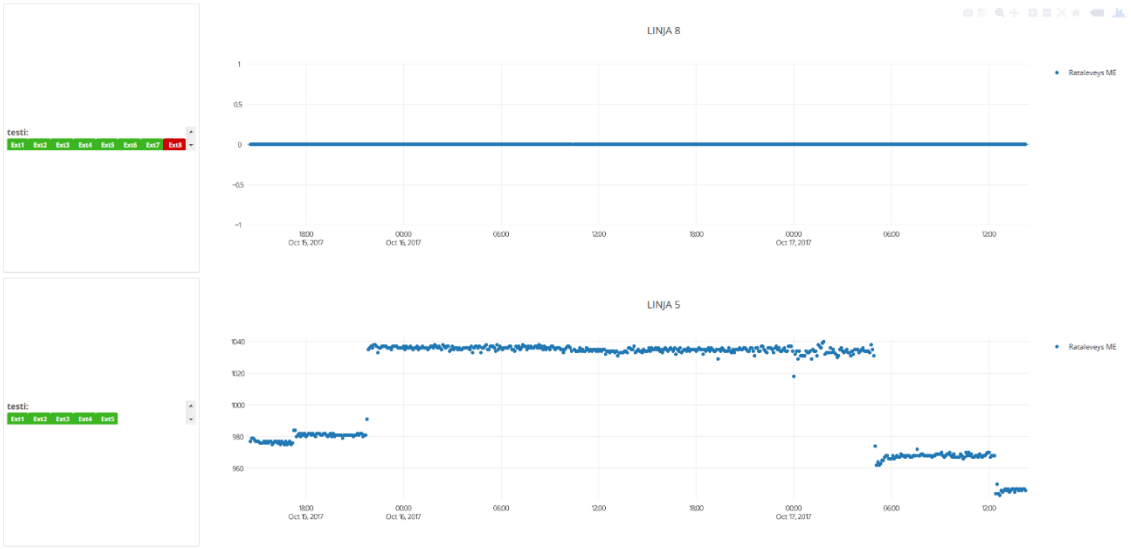
Ekstuurderikohtaisille vertailuarvoille sovellus luo vertailuarvon nimikkeellä rivin, johon sovellus generoi ekstruuderikohtaiset elementit, joiden nimikkeet osoittavat mille ekstruuderille elementti kuuluu. Yksinkertaisesti jos mitattu arvo on yli määritellyn rajan, elementti on punainen, kun normaalisti se on vihreä.

Lineaarisille vertailuarvoille sovellus luo samaan tapaan vertailuarvon nimikkeellä rivin ja rivillä on ”LIN” nimikkeellä varustettu elementti, joka on punainen tai vihreä poikkeamasta riippuen.

Toisin kuin ekstruuderin vertailuarvoilla, lineaarisille vertailuarvoille luodaan elementin viereen kenttä, jossa näytetään mittauksen viimeisin arvo. Tämä onnistuu lineaarisien vertailuarvojen kanssa koska yhdelle riville tulee aina vain yksi elementti, ja tilaa on siksi enemmän käytettävissä.

Jos jollekin vertailuarvolle ei ole laskettu onnistuneesti arvoa, sovellus ei generoi sitä arvoa vastaavaa elementtiä.

Reaaliaikanäkymäsivut on ohjelmoitu päivittämään itsensä automaattisesti tietyin aikavälein. Sovelluksen nykyisessä käyttökohteessa vaatimuksena oli korkeintaan muutaman minuutin välein päivittyminen.



Kuva 8 Täysi reaaliaikanäkymä

Kuvassa Kuva 8 on esitetty testikäytössä oleva lopullinen versio ohjelman täydestä reaaliaikanäkymästä.

4.7 Hallintasivut

Sovellukseen tarvittiin muutamia hallintasivuja sovelluksen eri osien määritysten lisäämiseen, päivittämiseen ja poistamiseen. Tätä varten sivustoon lisättiin kolme erillistä sivua, jotka käyttävät paljon samoja metodeja, jotka on erotettu omaan luokkaansa.

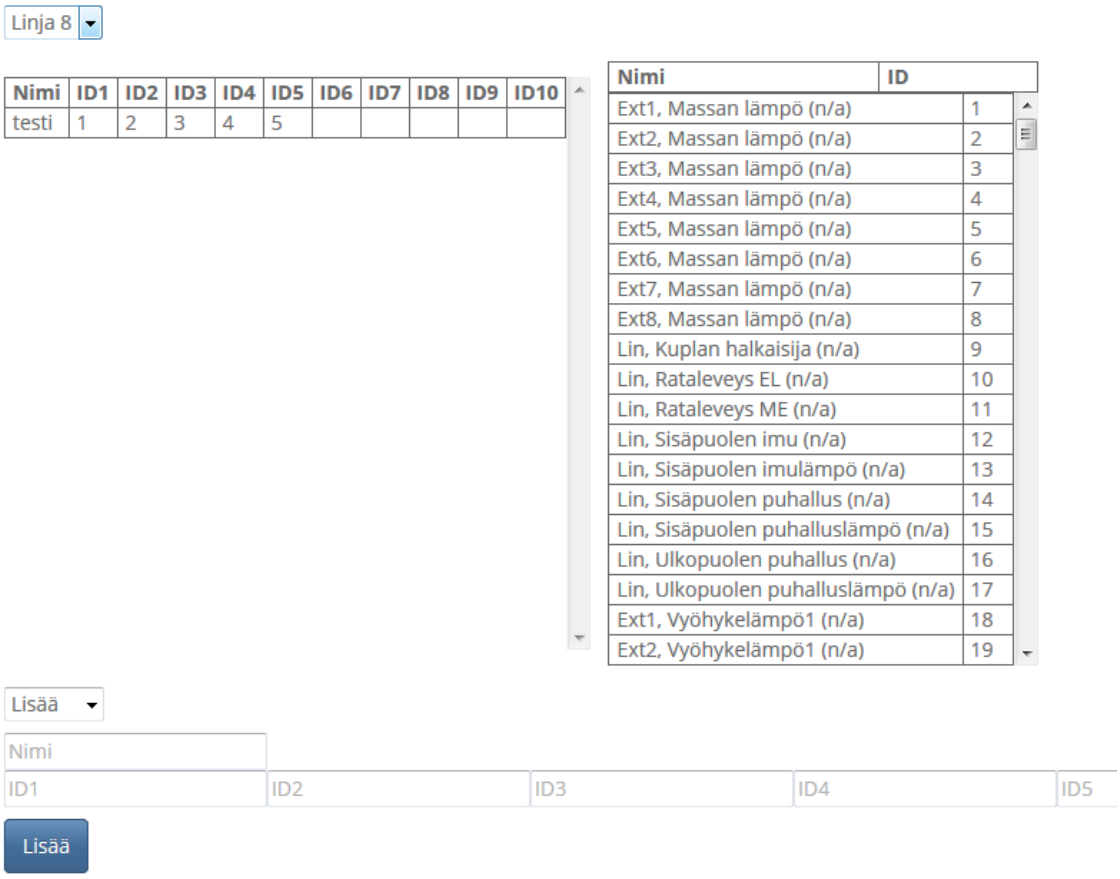
Hallintasivuilla on sivukohtaisesti esitetty ”GridView” elementillä muokattavana oleva taulukko. Kaikilla sivuilla on lisäksi pudotusvalikko, josta käyttäjä valitsee, haluaako hän lisätä tauluun uuden rivin, päivittää olemassa olevaa riviä vai poistaa jonkun rivin.

”GridView” elementti on ASP.NET viitekehyksen tarjoama elementti, johon voidaan suoraan sitoa datataulu. Elementin asetuksissa voidaan määritellä, halutaanko ruudukossa näyttää erillinen otsikkorivi ja datasidonnan laukaisemissa rivienluomistapahtumissa voidaan piilottaa haluttujen sarakkeiden kentät.

Pudotusvalikosta muokkaus moodia valittaessa sivut muuttavat sivulla näkyviä käyttöliittymäelementtejä sen mukaan mitä elementtejä valittu moodi käyttää. Esimerkiksi riviä poistettaessa ei tarvita tekstikenttiä arvojen syöttämiseksi vaan ainoaksi käyttäjäsyötteeksi riittää pudotusvalikko, josta käyttäjä valitsee poistettavan rivin.

Kaikissa tapauksissa hallintasivuilla halutaan muokata tietokannan taulusta riviä. Riveillä on eri hallintasivuista riippuen eri määriä kenttiä mihin määrityksiä voidaan tallentaa.

Jotta kaikki sivut voivat käyttää mahdollisimman paljon yhteisiä metodeja, sivuille luodaan tekstikentät rivin arvojen lisäämiseen tai päivittämiseen ohjelmallisesti. Tämä mahdollistaa lisäksi sovelluksen päivittämisen pienellä vaivalla, jos tulevaisuudessa halutaan muuttaa taulujen sarakkeiden määrää.



Kuva 9 Luokkienhallinta

Kuvassa Kuva 9 on esitetty esimerkkinä hallintasivusta lopullinen versio luokkienhallinnasta.

4.7.1 Ryhmien hallinta

Kuten nimestä voi päätellä, ryhmienhallintasivulla hallitaan linjaston tietokannan taulua, jossa on määritelty eri mittausdataryhmät, joita voidaan käyttää analyysi- tai luokka-analyysinäkymissä.

Taulun yhdellä rivillä on määritelty yksi mittausdataryhmä. Ryhmälle on annettu nimi omassa sarakkeessa ja maksimissaan kymmenen mittaukseen viittaavaa ID:tä.

Ryhmänlisäysmoodissa sovellus generoi tekstikentät nimelle ja kaikille ID:lle. Käyttäjä antaa haluamansa nimen ja haluamansa määrän ID:tä uudelle ryhmälle ja tallentaa ryhmän painamalla ”lisää” painiketta.

Tyhjäksi jätetyt ID kentät ovat sallittuja, eivätkä ne vaikuta sovelluksen toimintaan, kun dataryhmää käytetään jossain analyysinäkymässä. Jos ID kenttään on kuitenkin annettu kelvoton arvo, eli arvo joka ei ole kokonaisluku, sovellus ilmoittaa käyttäjävirheestä ponnahdusikkunalla.

Ryhmänpäivitysmoodissa sovellus näyttää samat tekstikentät kuin lisäysmoodissa, mutta lisäksi sovellus näyttää pudotusvalikon, jossa on kaikki tauluun tallennettujen ryhmien nimet.

Käyttäjä valitsee pudotusvalikolla haluamansa ryhmän, jolloin sovellus täyttää valitun ryhmän nimen ja ID:t vastaaviin tekstikenttiinsä. Käyttäjä voi muokata haluamiaan arvoja ja tallentaa päivitetyn ryhmän painamalla ”päivitä” painiketta, jolloin sovellus päivittää valittua ryhmää vastaavan rivin.

Ryhmänpoistomoodissa käyttäjä valitsee samasta pudotusvalikosta haluamansa ryhmän ja painamalla ”poista” painiketta sovellus poistaa ryhmää vastaavan rivin kokonaan tietokannan taulusta.

Ryhmienhallintanäkymässä on myös selite ”GridView” elementissä, johon on haettu ”IO_DEF” taulusta kaikki ID:t ja niitä vastaavat nimet. Selitteen tarkoitus on helpottaa käyttäjän toimintaa niin, ettei hänen tarvitse tietää ulkoa sovellukseen määriteltyjen mittauksien ID:tä.

4.7.2 Luokkien hallinta

Luokkien hallintanäkymässä käyttäjä voi muokata taulua, jossa on määritelty vertailuarvojen laskusäännöt, joita käytetään, kun analyysinäkymässä lasketaan uudet vertailuarvot ja kun reaaliaikanäkymässä muodostetaan mittauksien seurantaan käytetyt elementit.

Yksi ”luokka” on taulun rivi, jossa on määritelty luokan nimi, onko vertailu lineaarinen vai ei ja mitkä mittauksen kuuluvat luokkaan. Nykyisessä sovelluksen toteutuksessa yhdessä luokassa on maksimissaan kolme mittausta.

Luokanlisäysmoodissa sovellus generoi tekstikentät luokan nimelle ja mittauksille. Lisäksi sovelluksessa on pudotusvalikko, josta käyttäjä valitsee, onko lisättävä vertailuluokka lineaarinen vai ei.

Jos käyttäjä valitsee luokan lineaariseksi, lisättävien mittauksien tekstikentät vähennetään yhteen ja sovellus lisää tekstikentän K arvoa varten, joka on lineaarisessa vertailussa käytetty varmuuskerroin.

Koska mittaukset lisätään mittauksen nimen mukaan, sovellus ei tarkasta lisättyjä mittauksia vaan oletetaan, että käyttäjä tarkastaa lisätyt mittaukset, tai käyttäjä aikoo lisätä myöhemmin uuden mittauksen ”IO_DEF” tauluun.

Mittauksien lisäämisen helpottamiseksi luokkienhallinta näkymään on lisätty selite ”GridView” elementti, johon haetaan ”IO_DEF” taulusta kaikki uniikit käytössä olevat mittauksien nimet.

Samaan tapaan kuin ryhmienhallinnassa, päivitysmoodissa on samat elementit kuin lisäysmoodissa, mutta lisäksi päivitysmoodissa on pudotusvalikko, josta valitaan haluttu luokka. Valittua luokka vaihdettaessa sovellus päivittää asianmukaisiin kenttiin luokan arvot ja käyttäjä voi muokata haluamiaan arvoja.

Myös luokan poistaminen toimii samaan tapaan kuin ryhmienhallinnassa, eli käyttäjä vain valitsee haluamansa luokan ja painaa ”poista” painiketta.

4.7.3 Avainarvojen hallinta

Avainarvojen hallintanäkymässä käyttäjä voi muokata taulua, jossa on tallennettuna eri avainarvojen laskusäännöt. Yhdelle avainarvolle on taulun rivillä määritelty nimi, ID, käytetty laskusääntö ja laskentaan käytetyt ID:t sekä staattiset arvot K.

Lisäksi jokaista avainarvoa vastaa ”mittaus”, joka on määritelty taulussa ”IO_DEF” ja jossa avainarvolle on määritelty ID ja Block, joka kertoo mille ekstruuderille avainarvo kuuluu.

Lisäysmoodissa sovellus näyttää pudotus valikot, joista käyttäjä voi valita kuuluuko avainarvo jollekin ekstruuderille, vai onko se linjastolle yhteinen ja mitä laskusääntöä avainarvo käyttää.

Valitusta laskusäännöstä riippuen sovellus generoi tarpeellisen määrän ID ja K tekstikenttiä. Jos käyttäjä yrittää syöttää virheellisen arvon johonkin kentistä tai käyttäjä jättää jonkin arvon täyttämättä, sovellus ilmoittaa ponnahdusikkunalla käyttäjävirheestä.

Samaan tapaan kuin muissa hallintasivuissa, käyttäjä valitsee päivitys ja poistomoodeissa haluamansa avainarvon pudotusvalikosta ja päivitysmoodin ollessa käytössä sovellus lisää elementteihin valitun avainarvon arvot.

Jos käyttäjä muokkaa avainarvoa, sovellus päivittää avainarvotaulun lisäksi ”IO_DEF” taulua. Arvoja poistettaessa niitä vastaava rivi poistetaan myös ”IO_DEF” taulusta ja vastaavasti arvo lisättäessä arvolle luodaan uusi rivi ”IO_DEF” tauluun. Jos avainarvon ekstruuderia päivitetään, ”IO_DEF” tauluussa muokataan arvoa vastaavaa riviä.

Koska avainarvoon viittaava ID määritellään ”IO_DEF” taulussa ja ”IO_DEF” taulun ID:t ovat tietokannan generoimia automaattilukuja, uutta arvoa lisättäessä sovelluksen täytyy ensin luoda uusi rivi ”IO_DEF” tauluun, jotta avainarvotauluun saadaan asianmukainen ID.

4.8 Manuaalinen datapisteiden lisäys

Sovellukseen haluttiin mahdollisuus lisätä tiettyjä manuaalisesti lisättäviä mittauksia. Mittaukset olisivat esimerkiksi linjaston ajojen välissä linjaston telasta otettuja mittauksia.

Manuaaliseen datapisteiden lisäykseen luotiin yksinkertainen sivu, jossa on tekstikentät lisättävän mittauksen arvolle, työ- ja tuotetunnukselle, aikaleimalle ja lisääjän nimelle.

Sovellus täyttää tietokannasta saatujen tietojen perusteella automaattisesti viimeisimmän ajon työ- ja tuotetunnukset ja asettaa oletusarvoiseksi aikaleimaksi tämänhetkisen ajan sekuntien tarkkuudella. Käyttäjä voi halutessaan muokata näitä arvoja.

Jos käyttäjä yrittää lisätä virheellisen mittausarvon, sovellus ilmoittaa käyttäjävirheestä ponnahdusikkunalla.

Sovellus tarkastaa, onko käytettyjä työ- ja tuotetunnuksia olemassa vastaavissa tauluissa ja jos tunnukset ovat uusia, sovellus lisää tauluihin uudet tunnukset.

4.9 Tuotteen korjaus

Sovellukseen haluttiin mahdollisuus korjata tallennettujen mittausten tuotetunnukset valitun työn mukaan. Tarkoitus on tarjota käyttäjälle mahdollisuus korjata dataa tilanteessa, jossa työlle on asetettu vahingossa väärä tai väärinkirjoitettu tuotetunnus.

Tähän tarkoitukseen sovellukseen lisättiin sivu, jossa käyttäjä syöttää tekstikenttiin halutun työtunnuksen, jolle tuote korjataan, sekä korjattu tuotetunnus.

Jos korjattu tuotetunnus on entuudestaan käytetty, muokataan mittauksille, joilla on annettu työtunnus, korjattua tuotetta vastaava tuote ID. Jos korjattua tuotetunnusta ei ole olemassa, luodaan tietokantaan uusi tunnus ja sen ID muokataan halutuille mittauksille.

4.10 Viitekehyksen sivustolle yhteiset ominaisuudet

ASP.NET viitekehykseen kuuluu tiedostoja, joita sovellus käyttää kaikissa sivuissa.

4.10.1 Web.config

ASP.NET viitekehykseen kuuluu Web.config tiedosto, joka sisältää kaikki verkko asetukset, joita voidaan käyttää joko kaikissa palvelimen sovelluksissa, yksittäisessä sovelluksessa tai yksittäisissä sivuissa.

Tässä sovelluksessa tiedostoa käytetään koko sovelluksessa. Tässä kehityskohteessa Web.config tiedossa määriteltyjä asetuksia haluttiin muokata vain lisäämällä sovelluksen käyttämät Connection Stringit.

Connection Stringeillä tarkoitetaan linkityksiä, joissa yhdistetään yksinkertainen tekstimuotoinen nimi tietokantaan yhdistävään komentoon, kuten nähdään kohdassa Ohjelma 26.

Koska sovellus on vielä paikallisessa kehitysympäristössä, tietokantojen lähteet viittaavat paikallisiin tiedostoihin. Lopullisessa käyttökohteessa lähteinä on todennäköisesti verkko-osoitteen takana olevat tietokannat.

4.10.2 Site.Master

ASP.NET viitekehykseen kuuluu Site.Master sivu, jossa on määritelty kaikille sivuston sivuille yhteisiä elementtejä. Käytännössä sivuja ladattaessa sovellus lataa ensin Site.Masterissa määritellyn sisällön ja sen jälkeen sovellus lataa sivukohtaisen sisällön, joka esitetään Site.Masterin ”Content Placeholder” elementissä.

Site.Master sivussa on määritelty lähes kaikissa sivuissa näkyvä navigointipalkki, joka esitetään jatkuvasti ikkunan yläreunassa. Palkissa on linkit sivuston eri sivuille.

Lisäksi palkin vasemmassa laidassa on pieni kuvake, joka vie etusivulle ja oikeassa laidassa on kirjautumiselementti, joka vie kirjautumissivulle, jos käyttäjä ei ole kirjautunut sisään tai kirjaa käyttäjän ulos, jos käyttäjä on jo kirjautunut sisään.

Site.Master sivussa on myös määritelty lähes kaikissa sivuissa näkyvä alaviite, joka näytetään jatkuvasti ikkunan alareunassa. Alaviitteessä on sovelluksen tekijänoikeustiedot sekä logo, joka ohjaa sovelluksen valmistajan eli Denovo Oy:n sivuille.

Sekä navigointipalkki, että alaviite on asetettu paneelielementin sisään, sillä karsittuun reaaliaikanäkymään haluttiin näkyviin pelkästään linjastojen kuvaajat. Paneelielementtejä käyttämällä voidaan reaaliaikanäkymän koodissa määrittää elementit piilotetuiksi.

Site.Master sivun koodissa on lisäksi määritelty tietyille linkeille, että ne näytetään vain, jos käyttäjä on kirjautunut sisään. Sisäänkirjautumisesta on kerrottu tarkemmin luvussa 4.11 Käyttäjäkokemus.

4.11 Käyttäjäkokemus

Tässä luvussa kerrotaan tarkemmin, miten sovelluksen käyttäjäkokemusta on kehitetty ja miksi siihen liittyviä päätöksiä on tehty.

4.11.1 Aloitussivu

Kun käyttäjä menee sovelluksen sivulle selaimella, käyttäjä ohjataan aloitussivulle. Sivulla on sivuille yhteisten elementtien lisäksi sovelluksen logo ja sen alla kirjautumiselementti, joka vie kirjautumissivulle, jos käyttäjä ei ole kirjautunut sisään tai kirjaa käyttäjän ulos, jos käyttäjä on jo kirjautunut sisään.

Aloitussivulle ei tarvittu toteutuksen tässä vaiheessa juurikaan muuta sisältöä, mutta myöhemmin sivulle voidaan lisätä esimerkiksi asiakkaan määrittelemiä linkkejä tai yhteystietoja.

4.11.2 Töiden valinta

Töiden valinta haluttiin erotta analyysi- ja luokka-analyysinäkymissä omaan ponnahdusikkunaan. Ideana oli, että käyttäjä valitsee ensin tuotteen, sille haluamansa työt ja lopulta ajan jolta haetut mittaukset näytetään.

Käyttäjän ajatellaan haluavan vaihtaa valittuja töitä harvemmin, eli käyttäjä työt valittuaan todennäköisesti muokkaa haun aikahaarukkaa moneen kertaan saadakseen haluamanlaisensa jakson.

Tästä syystä töiden valinta näkymää ei haluttu jatkuvasti näkyville, joten se päätettiin lisätä omana ponnahdusikkunanaan. Tämä mahdollisti myös ponnahdusikkunan lisäämisen sekä analyysi- että luokka-analyysinäkymiin pienellä vaivalla tekemällä siitä User Controllin.

4.11.3 Virhe ponnahdusikkuna ja poikkeuksien käsittely

Sovellukseen on luotu oma User Control virhe ponnahdusikkunaa varten. Tämä johtuu siitä, että kaikilla sivuilla voidaan mahdollisesti tarvita virheestä ilmoittavaa ponnahdusikkunaa.

Aikaisemmin mainittujen ennakoitujen virhetilanteiden lisäksi ponnahdusikkunaa tarvitaan ilmoittamaan odottamattomista virheistä. ASP.NET viitekehyksen palvelinkoodissa käyttämä C# kieli tarjoaa ”Try Catch” rakenteen, jolla ohjelma voi ennakoida virheitä ja selvitä niistä niin, ettei koko sovellus kaadu.

Rakennetta voidaan käyttää havaitsemaan poikkeuksia, jolloin sovellus ilmoittaa odottamattomasta poikkeuksesta käyttäjälle ponnahdusikkunalla ja kirjaa tarkemmat tiedot poikkeuksesta sovelluksen loki tiedostoon.

Kaikista virheistä ilmoitetaan samalla ponnahdusikkunalla. Kohdatusta virheestä riippuen sovellus lisää ponnahdusikkunaan asianmukaisen virheviestin.

4.11.4 Sisäänkirjautuminen

Sovelluksessa käyttäjä voi kirjautua, jos hänellä on tiedossa tietokannassa asetetut järjestelmän ylläpitäjän kirjautumistiedot. Sovelluksessa ei ole tässä toteutuksessa varsinaisia käyttäjätilejä, vaan vain yksi tunnus, jolla kirjautumalla käyttäjä saa käyttöönsä täydet toiminnallisuudet.

Luokka-analyysinäkymä, kaikki vertailuarvojen laskennan toiminnallisuudet, hallintasivut ja tuotteen korjaus ovat käytettävissä vain, jos käyttäjä on kirjautunut sisään.

Site.Master sivussa on määritetty, että linkit, jotka ohjaavat vain kirjautuneille käyttäjille tarkoitetuille sivuille, ovat piilotettu navigointipalkista, jos käyttäjä ei ole kirjautunut sisään.

Lisäksi jokaista rajoitettua sivua ladattaessa sovellus tarkastaa onko käyttäjä kirjautunut, ja ohjaa käyttäjän automaattisesti kirjautumissivulle, eli rajoitettuihin toiminnallisuuksiin ei pääse sisään kirjautumatta vain käyttämällä rajoitettujen sivujen verkko-osoitetta.

Kirjautumissivu koostuu yksinkertaisesti tekstikentistä käyttäjätunnusta ja salasanaa varten, sekä painikkeesta, jolla syötetään kirjautumistiedot. Sivu ilmoittaa virheellisistä kirjautumistiedoista tai onnistuneesta kirjautumisesta.

5. JATKOKEHITYS

Kehitetylle sovellukselle on jo jatkokehityssuunnitelmia ja osaa näistä kehityssuunnista on jo aloitettu seurata tämän kirjoittamisen aikaan.

5.1 Data Logger ja OPC

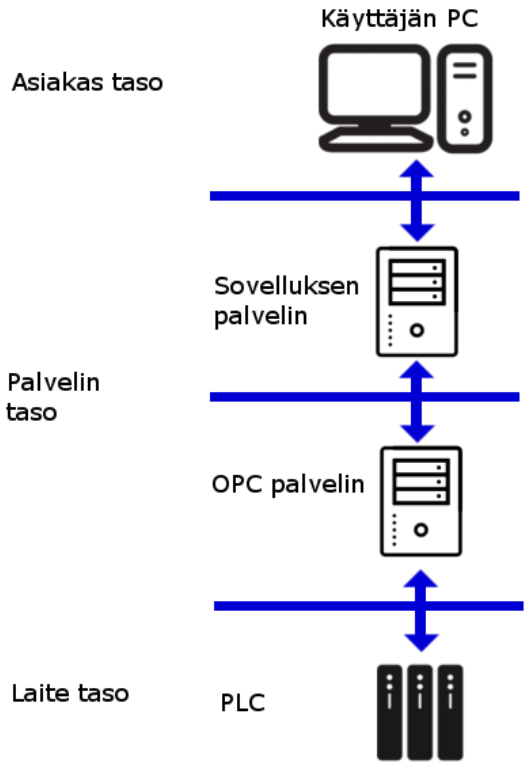
Tässä työssä kerrottiin tarkemmin vain sovelluksen toiminnallisuuksista, jotka liittyvät työn aiheeseen, eli historiadatan hyödyntämiseen. Sovellukseen kuluu kuitenkin myös niin sanottu Data Logger toiminnallisuus.

Linjaston logiikan, jonka dataan päästään vain valvomon tietokoneen kautta, mittaukset Data Logger saa suoraan tekstitiedostosta, johon ne on kirjoitettu kappaleessa 2.3 Toteutusympäristö kuvatulla tavalla. Tekstitiedostossa on yhdellä rivillä mittauksen ID ja sen arvo, jotka on erotettu välimerkillä.

Linjaston logiikan, joka on suoraan kiinni laitoksen verkossa, ja Data Loggerin välissä arkkitehtuurissa on OPC palvelin. OPC Foundation tarjoaa ilmaisen .NET kirjaston OPC palvelimen kanssa kommunikoimiseen, jota hyödyntämällä sovellus voi lukea linjaton mittausarvoja.

OPC koostuu asiakkaasta ja palvelimesta. Tässä käyttötapauksessa OPC-asiakas on kehitettävä sovellus ja käytettäväksi OPC-palvelimeksi valittiin KEPServerEX, joka on Kepwaren tarjoama OPC-palvelinsovellus. Kirjoittamisen hetkellä KEPServerEX ohjelmasta oli käytössä versio 6.3.

OPC-palvelin lukee linjaston logiikoilta mittausarvoja ja välittää niitä eteenpäin asiakkaalle. Arkkitehtuuri on esitetty kuvassa Kuva 10.



Kuva 10 verkon arkkitehtuuri

OPC palvelin voisi olla asennettuna erillisellä palvelinkoneella, josta Data Logger lukisi datat Ethernetin yli, mutta tässä käyttötapauksessa OPC palvelin päätettiin asentaa samalle palvelinkoneelle kuin sovellus.

OPC:n avoimesta luonteesta johtuen palvelinsovelluksena voitaisiin käyttää muutakin sovellusta, mutta KEPServerEX soveltuu erityisen hyvin kehityskäyttöön, sillä se tarjoaa rajallista toiminnallisuutta ilmaiseksi. Lisäksi KEPServerEX mahdollistaa mittauksien simuloinnin, eikä kehityksen ja testauksen aikana tarvita logiikkaa, johon OPC-palvelin olisi yhteydessä.

OPC on OPC Foundationin kehittämä ja ylläpitämä teollisuusautomaation yhteistoimivuus standardi [15]. OPC on avoin standardi, mutta monet OPC Foundationin välittämät sovellukset ja kehitystyökalut ovat saatavilla vain säätiön jäsenille ja yritysjäsenille. Tämä tarkoittaa sitä, että osa OPC materiaaleista on maksumuurin takana, mikä voi vaikuttaa jatkokehitykseen OPC:ta käytettäessä.

OPC on yleisesti automaatiossa käytetty teknologia, joten sovelluksen kehittäminen sitä hyödyntäväksi lisää sen yleishyödyllisyyttä.

5.2 Yleishyödyllisyys

Sovellus on alun perin suunniteltu nimenomaan muoviteollisuuden käyttöön, joka lähinnä näkyy käytetyistä laskumenetelmistä sekä sovelluksen käyttökohteen sanelemista tietokantojen määritelmistä.

Sovellusta haluttaisiin kuitenkin jatkossa kehittää suuntaan, jossa sitä voitaisiin käyttää myös muiden teollisuudenalojen kohteissa.

Sovellusta on jo nyt kehitetty toimimaan mahdollisimman itsenäisesti käyttökohteen sanelemista määristä riippumatta. Esimerkiksi ekstruudereiden tai linjastojen määrä ei ole sovelluksen toteutuksessa kiveen hakattu, vaikka niiden määrä on ollut tiedossa koko sovelluksen kehityksen ajan.

Laskumenetelmiä varten sovellukseen voitaisiin kehittää tulevaisuudessa mahdollisuus käyttäjälle lisätä omia laskusääntöjä. Nykyinen avainarvojen laskennan toteutus tukee perustasolla tällaista toiminnallisuutta, mutta laskusääntöjen luomiseen täytyisi luoda käyttöliittymä, joka olisi todennäköisesti erittäin monimutkainen.

Myös tietokantojen rakenteita tulee tulevaisuudessa miettiä tarkoin. Tähänkin asti tietokantojen rakennetta on muutetta alkuperäisistä annetuista muotoiluista kehityksen aikana, mutta etenkin Microsoft Accessista SQL:ään siirtymistä tulisi harkita ja ottaa selvää tämän muutoksen tarjoamista mahdollisuuksista.

6. YHTEENVETO

Historiadatan hyödyntämisellä prosessiteollisuudessa on selkeitä prosessin monitorointiin tuotettuja lisähyötyjä. Yhdistämällä prosessista tehokkaasti kerätty mittausdata sopivaan sovellukseen, voidaan prosessin valvontaa tehostaa luomalla käyttäjälle yksinkertaisempia indikaattoreita prosessin tilasta.

Sovellus siis mahdollistaa prosessin tehokkaan valvomisen vähemmällä vaaditulla käyttäjän osaamisella. Tietyissä prosesseissa voitaisiin harkita valvonnan täyttä automatisointia sovelluksen avulla.

Sovellus kuitenkin vaatii osaavan käyttäjän valitsemaan sopivia datan jaksoja vertailuarvojen luomista varten. Prosessikohtaisella asiantuntemuksella sovellus voitaisiin kehittää havaitsemaan automaattisesti sopivia jaksoja. Kuitenkin toive sovelluksen jatkokehityksestä yleishyödyllisempään suuntaan tarkoittaa sitä, ettei tällaisia kehityssuuntia tulla seuraamaan.

Sovelluksen kehittäminen nimenomaan verkkosovellukseksi alusta alkaen hyödyttää organisaatioita, joilla on mahdollisesti tarpeita prosessin etätarkkailuun. Verkkosovellus myös mukailee nykyisin hyvin vahvoilla olevaa esineiden internet trendiä, joka on myös prosessiteollisuudessa pinnalla.

Lisäksi verkkosovelluksen kehittämistä varten valittu viitekehys tarjoaa mahdollisuuden luoda yksinkertainen, graafinen käyttöliittymä. Tällainen käyttöliittymä tukee toivetta tarjota helposti ymmärrettävä käyttäjäkokemus vähemmän asiantunteville käyttäjille.

LÄHTEET

[1] M.B. Karan, A. Ulucan, M. Kaya, Credit risk estimation using payment history data: a comparative study of Turkish retail stores, Central European Journal of Operations Research, Vol. 21, No. 2, 2013, pp. 479-494.

[2] V.W. Zheng, Y. Zheng, X. Xie, Q. Yang, Towards mobile intelligence: Learning from GPS history data for collaborative recommendation, Artificial Intelligence, Vol. 184-185, 2012, pp. 17-37.

[3] Rapid evaluation of operation performance of multi-chiller system based on history data analysis, EvaluationMulti-chiller systemOperation performanceData analysis, 2017, pp. 162-170.

[4] J. Gao, R. Sion, S. Lederer, Collaborative location certification for sensor networks, ACM Transactions on Sensor Networks (TOSN), Vol. 6, No. 4, 2010, pp. 1-26.

[5] C. Schwenke, V. Vasyutynskyy, A. Roder, K. Kabitzsch, Analysis of maintenance histories of industrial equipment with frequent maintenance demand, pp. 299-304.

[6] M. Chen, S. Mao, Y. Liu, Big Data: A Survey, Mobile Networks and Applications, Vol. 19, No. 2, 2014, pp. 171-209.

[7] F. Xia, L.T. Yang, L. Wang, A. Vinel, Internet of Things, International Journal of Communication Systems, Vol. 25, No. 9, 2012, pp. 1101-1102.

[8] Blow film line, Reifenhäuser, web page. Available (accessed 15.11.2017): http://www.reifenhauser.com/.

[9] R. Sinclair, SpringerLink (Online service), I. Books24x7, From Access to SQL Server, Apress, Berkeley, CA, 2000, .

[10] ASP.NET overview, Microsoft, web page. Available (accessed 25.9.2017): https://docs.microsoft.com/fi-fi/aspnet/overview.

[11] W. Penberthy, Beginning ASP. NET for Visual Studio 2015, 1st ed. Wrox, Hoboken, 2016, .

[12] What is plotly.js? Plotly, web page. Available (accessed 25.9.2017): https://plot.ly/javascript/.

[13] Data-Driven Documents, web page. Available (accessed 25.9.2017): https://d3js.org/.

[14] Stack.gl, web page. Available (accessed 25.9.2017): http://stack.gl/.

[15] What is OPC? OPC Foundation, web page. Available (accessed 27.9.2017): https://opcfoundation.org/about/what-is-opc/.

LIITE A: OHJELMAT

```
DataTable arvot = DByhteys.HaeTauluNoPar("SELECT Description,
ID, Block, Block_Table, Unit FROM IO_DEF ORDER BY ID ASC",
Linjat.SelectedValue);
```

Ohjelma 1 Esimerkki OleDb kyselystä

```
IOlista.Items.Clear();
IOlista.DataSource = arvot;
IOlista.DataValueField = "ID";
IOlista.DataTextField = "Description";
IOlista.DataBind();
```

Ohjelma 2 Esimerkki datan sitomisesta elementtiin

```
DataTable arvot = DByhteys.HaeTauluNoPar("SELECT DISTINCT Description FROM IO_DEF WHERE
Block_Table <> 'LIN' AND (Block_Table <> 'KEY' OR Block <> 0)", Linjat.SelectedValue);

Mittaukset.Items.Clear();
Mittaukset.DataSource = arvot;
Mittaukset.DataValueField = "Description";
Mittaukset.DataTextField = "Description";
Mittaukset.DataBind();
Mittaukset.Items.Add(new ListItem("Lin","LIN"));
```

Ohjelma 3 Mittaustyyppien hakeminen ja sitomien pudotusvalikkoon ”Mittaukset”

```
OleDbCommand cmd = new OleDbCommand();
StringBuilder cnn = new StringBuilder();

if (Mittaukset.SelectedValue == "LIN")
{
            cnn.Append("SELECT Description, ID, Block, Block_Table, Unit FROM IO_DEF WHERE
            Block_Table = 'LIN' OR (Block_Table = 'KEY' AND Block = 0)");
}
else
{
            cnn.Append("SELECT Description, ID, Block, Block_Table, Unit FROM IO_DEF WHERE
            Description = @desc");
             cmd.Parameters.AddWithValue("@desc", Mittaukset.SelectedValue);
}

cnn.Append(" ORDER BY ID ASC");
cmd.CommandText = cnn.ToString();

DataTable arvot = DByhteys.HaeTaulu(cmd, connStr);
```

Ohjelma 4 Näytettävät mittaukset mittaustyypin mukaan

```
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = "SELECT * FROM Ryhmät WHERE ID = @id";
cmd.Parameters.AddWithValue("@id", Ryhmät.SelectedValue);
DataTable ValittuRyhmä = DByhteys.HaeTaulu(cmd, connStr);

int sarakkeet = ValittuRyhmä.Columns.Count;

List<int> RyhmänIDt = new List<int>();

for(int i = 2; i < sarakkeet; i++)
{
	if (ValittuRyhmä.Rows[0][i] != DBNull.Value)
	{
		RyhmänIDt.Add(int.Parse(ValittuRyhmä.Rows[0][i].ToString()));
	}
}

StringBuilder cnn = new StringBuilder();
cmd.Parameters.Clear();

cnn.Append("SELECT Description, ID, Block, Block_Table, Unit FROM IO_DEF WHERE");

for(int j = 0; j < RyhmänIDt.Count; j++)
{
	cnn.Append(" ID = @id" + j.ToString() + " OR");
	cmd.Parameters.AddWithValue("@id" + j.ToString(), RyhmänIDt[j]);
}

if (cnn.ToString().EndsWith("OR"))
{
	cnn.Length += -2;
}

cnn.Append(" ORDER BY ID ASC");

if (RyhmänIDt.Count == 0)
{
	cnn.Clear();
	cnn.Append("SELECT Description, ID, Block, Block_Table, Unit FROM IO_DEF WHERE 1 = 0");
}

cmd.CommandText = cnn.ToString();
DataTable arvot = DByhteys.HaeTaulu(cmd, connStr);
```

Ohjelma 5 Ryhmän mittauksien haku

```
for (int i = 0; i < arvot.Rows.Count; i++)
            {
              if (arvot.Rows[i][3].ToString() == "LIN" || (arvot.Rows[i][3].ToString() ==
              "KEY" && arvot.Rows[i][2].ToString() == "0"))
                {
                    arvot.Rows[i][0] = "Lin, " + arvot.Rows[i][0].ToString();
                }
                else
                {
                    string str;

                    if (int.Parse(arvot.Rows[i][2].ToString()) < 10)
                    {
                        str = arvot.Rows[i][2].ToString();
                    }
                    else
                    {
                        str = (int.Parse(arvot.Rows[i][2].ToString()) - 10).ToString();
                    }

                    arvot.Rows[i][0] = "Ext" + str + ", " + arvot.Rows[i][0].ToString();
                }

                string unit;

                if(!string.IsNullOrEmpty(arvot.Rows[i][4].ToString()))
                {
                    unit = arvot.Rows[i][4].ToString();
                }
                else
                {
                    unit = "n/a";
                }

                arvot.Rows[i][0] = arvot.Rows[i][0].ToString() + " (" + unit + ")";
            }
```

Ohjelma 6 Mittauksien kuvaukset täydentävä silmukka

```
DataTable tuotteet = DByhteys.HaeTauluNoPar("SELECT ID, Tunnus FROM Tuotteet",
Linjat.SelectedValue);

Tuotehaku.Items.Clear();
Tuotehaku.Items.Insert(0, new ListItem("valitse tuote", "noSelection"));
Tuotehaku.DataSource = tuotteet;
Tuotehaku.DataValueField = "ID";
Tuotehaku.DataTextField = "Tunnus";
Tuotehaku.DataBind();
```

Ohjelma 7 Datan sidonta tuotehaun pudotusvalikkoon

```
TyötAika.Text = TyötPVM.SelectedDate.ToString("dd.MM.yyyy");
```

Ohjelma 8 Päivämäärä kalenterista tekstikenttään

```
DropDownList Tuotehaku = (DropDownList)Parent.FindControl("Tuotehaku");
DropDownList Linjat = (DropDownList)Parent.FindControl("Linjat");

Työt.Items.Clear();

string AlkuDate = TyötAlkuAika.Text;
DateTime AlkuDt;
string LoppuDate = TyötLoppuAika.Text;
DateTime LoppuDt;

if ((DateTime.TryParse(AlkuDate, out AlkuDt) || string.IsNullOrEmpty(AlkuDate)) &&
(DateTime.TryParse(LoppuDate, out LoppuDt) || string.IsNullOrEmpty(LoppuDate)))
{
       OleDbCommand cmd = new OleDbCommand();
       StringBuilder cnn = new StringBuilder();

       cnn.Append("SELECT ID, Tyonumero FROM Tilaukset WHERE");

       if (Tuotehaku.SelectedIndex != 0)
       {
              cnn.Append(" Tuote = @tuote AND");
              cmd.Parameters.AddWithValue("@tuote", Tuotehaku.SelectedItem.Text);
       }

       if (!string.IsNullOrEmpty(AlkuDate))
       {
              cnn.Append(" Valmistui >= @alku AND");
              cmd.Parameters.AddWithValue("@alku", AlkuDt);
       }

       if (!string.IsNullOrEmpty(LoppuDate))
       {
              cnn.Append(" Valmistui <= @loppu");
              cmd.Parameters.AddWithValue("@loppu",LoppuDt);
       }

       if (cnn.ToString().EndsWith("AND"))
       {
              cnn.Length += -3;
       }
       else if (cnn.ToString().EndsWith("WHERE"))
       {
              cnn.Length += -5;
       }

       cmd.CommandText = cnn.ToString();
       DataTable työt = DByhteys.HaeTaulu(cmd, Linjat.SelectedValue);

       Työt.DataSource = työt;
       Työt.DataValueField = "ID";
       Työt.DataTextField = "Tyonumero";
       Työt.DataBind();

}
```

Ohjelma 9 Töiden haku tietokannasta

```
<p>
	<b>Linja   </b>
	<asp:DropDownList ID="Linjat" runat="server" autopostback="true"
	OnSelectedIndexChanged="Linjat_SelectedIndexChanged">
		<asp:ListItem Value="Linja1" Text="Linja 1" />
		<asp:ListItem Value="Linja2" Text="Linja 2" />
	</asp:DropDownList>
</p>
```

Ohjelma 10 Linjaston valinta pudotusvalikko

```
data.AxisLabels.Clear();
data.IDt.Clear();

if (data.Korrelaatio == true)
{
	data.AxisLabels.Add("");
	data.IDt.Add(int.Parse(IOlista2.SelectedValue));
}

for(int i = 0; i < IOlista.Items.Count; i++)
{
	if (IOlista.Items[i].Selected)
	{
		if (!(data.Korrelaatio == true) || ((data.Korrelaatio == true) &&
		(int.Parse(IOlista.Items[i].Value) != data.IDt[0])))
		{
			data.AxisLabels.Add(IOlista.Items[i].Text);
			data.IDt.Add(int.Parse(IOlista.Items[i].Value));
		}
	}
}
```

Ohjelma 11 Nimikkeiden ja ID:den lisäys

```
OleDbCommand cmd = new OleDbCommand();
StringBuilder cnn = new StringBuilder();

cnn.Append("SELECT Block, BlockNum, Block_Table, ID FROM IO_DEF WHERE ");

if (data.Korrelaatio == true)
{
	cnn.Append("ID = @fid OR ");
	cmd.Parameters.AddWithValue("@fid", IOlista2.SelectedValue);
}

for (int j = 0; j < data.IDt.Count; j++)
{
	if (data.Korrelaatio == true)
	{
		if (IOlista2.SelectedValue != data.IDt[j].ToString())
		{
			cnn.Append("ID = @id" + j.ToString() + " OR ");
			cmd.Parameters.AddWithValue("@id" + j.ToString(), data.IDt[j]);
		}
	}
	else
	{
		cnn.Append("ID = @id" + j.ToString() + " OR ");
		cmd.Parameters.AddWithValue("@id" + j.ToString(), data.IDt[j]);
	}
}

if (cnn.ToString().EndsWith("OR "))
{
	cnn.Length += -3;
}

cmd.CommandText = cnn.ToString();
DataTable IDtaulu = DByhteys.HaeTaulu(cmd, Linjat.SelectedValue);

DataTable IDjärjestetty = AnalyysiDatat.JärjestäTaulu(data.IDt, IDtaulu);
```

Ohjelma 12 Mittauksien tietojen hakeminen

```
public static DataTable JärjestäTaulu(List<int> IDlista, DataTable IDtaulu)
{
	if (IDtaulu.Rows.Count != 0)
	{
		var query =
		from b in IDlista
		join r in IDtaulu.AsEnumerable() on b equals r.Field<int>("ID")
		select r;

		return query.CopyToDataTable();
	}
	else
	{
		return new DataTable();
	}
}
```

Ohjelma 13 Datataulun järjestäminen ID:n mukaan

```
cnn.Clear();
cmd.Parameters.Clear();

cnn.Append("SELECT DISTINCT a.Stamp FROM EX_PV_1 AS a WHERE a.Block = 1");

cmd.CommandText = cnn.ToString();
DataTable taulu = DByhteys.HaeTaulu(CmdEhto.PnWIDnDate(cmd, Työt, Tuotehaku.SelectedValue,
RefAlku, RefLoppu), Linjat.SelectedValue);

data.määrä = IDjärjestetty.Rows.Count;

string datestring;

for (int j = 0; j < taulu.Rows.Count; j++)
{
	DateTime date = (DateTime)taulu.Rows[j][0];datestring =
	date.ToString("yyyy-MM-dd HH:mm:ss");

	data.Tdata.Add(datestring);
}

List<DataStruct> DataLista = new List<DataStruct>();

AnalyysiDatat.TäytäSuperlista(false, data.YSuperList, DataLista, IDjärjestetty,
Linjat.SelectedValue, Työt, Tuotehaku.SelectedValue, RefAlku, RefLoppu);
```

Ohjelma 14 Aikaleimojen ja mittausdatan lisäys

```
if (data.Manual)
{
        cnn.Clear();
        cmd.Parameters.Clear();

        cnn.Append("SELECT a.Data, a.Stamp FROM LAB_PV_1 AS a WHERE 1 = 1");

        cmd.CommandText = cnn.ToString();
        taulu = DByhteys.HaeTaulu(CmdEhto.PnWIDnDate(cmd, Työt, Tuotehaku.SelectedValue,
        RefAlku, RefLoppu),Linjat.SelectedValue);

        for (int k = 0; k < taulu.Rows.Count; k++)
        {
                DateTime date = (DateTime)taulu.Rows[k][1];
                datestring = date.ToString("yyyy-MM-dd HH:mm:ss");

                data.Ymanual.Add((float)taulu.Rows[k][0]);
                data.Xmanual.Add(datestring);
        }
}

data.ConvertList();
```

Ohjelma 15 Manuaalisen datan lisäys

```
JavaScriptSerializer javaSerial = new JavaScriptSerializer();

if (Korrelaatio == false)
{
        Xconverted = javaSerial.Serialize(Tdata);

        YConverted = javaSerial.Serialize(YSuperList);

        if(Manual)
                  {
                       YMConverted = javaSerial.Serialize(Ymanual);
                       XMConverted = javaSerial.Serialize(Xmanual);
                  }

}
else
{
        Xconverted = javaSerial.Serialize(YSuperList[0]);

        YConverted = javaSerial.Serialize(YSuperList);
}

LabelsConverted = javaSerial.Serialize(AxisLabels);

Tdata.Clear();
YSuperList.Clear();

Ymanual.Clear();
Xmanual.Clear();
```

Ohjelma 16 Mittausdatan kääntäminen

```
var d3 = Plotly.d3;
var WIDTH_IN_PERCENT_OF_PARENT = 100,
    HEIGHT_IN_PERCENT_OF_PARENT = 90;

var gd3 = d3.select("div[id='analyysi_graafi']")
    .style({
        width: WIDTH_IN_PERCENT_OF_PARENT + '%',
        'margin-left': (100 - WIDTH_IN_PERCENT_OF_PARENT) / 2 + '%',
        height: HEIGHT_IN_PERCENT_OF_PARENT + 'vh',
        'margin-top': (100 - HEIGHT_IN_PERCENT_OF_PARENT) / 2 + 'vh'
    });

var analyysi_graafi = gd3.node();

window.onresize = function () { Plotly.Plots.resize(analyysi_graafi) };
```

Ohjelma 17 Kuvaajan sovittelu

```
X = <%= this.data.Xconverted %>;

DatanMäärä = <%= this.data.määrä%>;

var Labels = <%= this.data.LabelsConverted%>;

var Traces = [];

Data = <%= this.data.YConverted%>;

var alku;

<% if(this.data.Korrelaatio == false) { %>
alku = 0;
<% } else { %>
    alku = 1;
<% } %>

    for (i = alku; i < DatanMäärä; i++) {
        var Y = Data[i];
        var Label = Labels[i];
        var Trace = {
            x: X,
            y: Y,
            mode: 'markers',
            type: 'scattergl',
            name: Label
        }
        Traces.push(Trace);
    }

<% if(this.data.Manual == true) { %>
var Y = <%= this.data.YMConverted %>;
var X = <%= this.data.XMConverted %>;
    var Trace = {
        x: X,
        y: Y,
        mode: 'markers',
        type: 'scattergl',
        name: 'Manuaalit'
    }
    Traces.push(Trace)
<% } %>

    var layout = {
        showlegend: true
    };

Plotly.plot(analyysi_graafi, Traces, layout);
```

Ohjelma 18 Kuvaajan muodostaminen

```
<%@ Register TagPrefix="uc" TagName="Työt" Src="~/UserControls/TyötPopup.ascx" %>

<uc:Työt id="Työt" Runat="server" />
```

Ohjelma 19 User Controllin käyttäminen

```
data.Xref = 0;
data.Yref = 0;
data.Zref = 0;

data.meancount = 0;

for (int i = 0; i < DataListat[0].Count; i++)
{
	float arvox = 0;
	float arvoy = 0;
	float arvoz = 0;

	try
	{
		arvox = DataListat[0][i];
		arvoy = DataListat[1][i];
		arvoz = DataListat[2][i];
	}
	catch
	{

	}
	finally
	{
		data.Xref += arvox;
		data.Yref += arvoy;
		data.Zref += arvoz;
		data.meancount += 1;
	}
}

data.Xref = data.Xref / data.meancount;
data.Yref = data.Yref / data.meancount;
data.Zref = data.Zref / data.meancount;
```

Ohjelma 20 Keskiarvon laskenta

```
var trace3 = {
                x: X1,
                y: Y1,
                z: Z1,
                mode: 'markers',
                marker: {
                    size: 5,
                    opacity: 1,
                    color: F,
                    colorscale: 'Jet',
                    colorbar: {
                        title: Flabel,
                    },
                },
                type: 'scatter3d',
                name: 'Joukko 1',
            };
```

Ohjelma 21 Väriskaalamoodin datajoukon muodostaminen

```
public static DataTable HaeTaulu(OleDbCommand cmd, string connStr)
{
	var table = new DataTable();

	using(var da = new OleDbDataAdapter())
	{
		var myCon = new OleDbConnection(
		ConfigurationManager.ConnectionStrings[connStr].ConnectionString);
		cmd.Connection = myCon;
		da.SelectCommand = cmd;
		da.Fill(table);
		da.Dispose();
		myCon.Dispose();
	}

	return table;
}
```

Ohjelma 22 Datataulun hakeva metodi

```
public static void LähetäTauluun(OleDbCommand cmd, string connStr)
{
	using (var myCon = new
	(ConfigurationManager.ConnectionStrings[connStr].ConnectionString))
	{
		cmd.Connection = myCon;
		myCon.Open();
		cmd.ExecuteNonQuery();
		myCon.Close();
		myCon.Dispose();
	}
}
```

Ohjelma 23 Tietokantaa muokkaava metodi

```
OleDbCommand cmd = new OleDbCommand();

DataTable taulu;

if (meas)
{
	taulu = DByhteys.HaeTaulu(CmdEhto.Meas(IDjärjestetty, j), connStr);
}
else
{
	taulu = DByhteys.HaeTaulu(CmdEhto.PnWIDnDate(CmdEhto.Data(IDjärjestetty, j), Työt,
	PID, RefAlku, RefLoppu), connStr);
}

var lista = new List<float>();

SuperLista.Add(lista);

float f = 0;

for (int k = 0; k < taulu.Rows.Count; k++)
{
	if (taulu.Rows[k][1] != DBNull.Value)
	{
		f = (float)taulu.Rows[k][1];
	}

	lista.Add(f);
}

return lista;
```

Ohjelma 24 Mittausarvojen lisääminen listaan

```
StringBuilder cnn = new StringBuilder().Append(cmd.CommandText);

bool WIDinUse = false;

if (PID != "noSelection")
{
        cnn.Append(" AND a.PID = @pid");
        cmd.Parameters.AddWithValue("@pid",PID);
}

if (Työt.SelectedIndex != -1)
{
        cnn.Append(" AND (");
        WIDinUse = true;
}

for (int i = 0; i < Työt.Items.Count; i++)
{
        if (Työt.Items[i].Selected)
        {
                cnn.Append(" a.WID = @wid" + i.ToString() + " OR");
                cmd.Parameters.AddWithValue("@wid" + i.ToString(), Työt.Items[i].Value);
        }
}

if (cnn.ToString().EndsWith("OR"))
{
                cnn.Length += -2;
}

if (WIDinUse == true)
{
                cnn.Append(")");
}

cnn.Append(" AND");

string AlkuDate = Alku.Text;
string LoppuDate = Loppu.Text;
DateTime AlkuDt;
DateTime LoppuDt;

if ((DateTime.TryParse(AlkuDate, out AlkuDt) || string.IsNullOrEmpty(AlkuDate)) &&
(DateTime.TryParse(LoppuDate, out LoppuDt) || string.IsNullOrEmpty(LoppuDate)))
{
        if (!string.IsNullOrEmpty(AlkuDate))
        {
                cnn.Append(" a.Stamp >= @alku AND");
                cmd.Parameters.AddWithValue("@alku", AlkuDt);
        }

        if (!string.IsNullOrEmpty(LoppuDate))
        {
                cnn.Append(" a.Stamp <= @loppu");
                cmd.Parameters.AddWithValue("@loppu", LoppuDt);
        }
}

if (cnn.ToString().EndsWith("AND"))
{
        cnn.Length += -3;
}

cnn.Append(" ORDER BY a.Stamp ASC");

cmd.CommandText = cnn.ToString();

return cmd;
```

Ohjelma 25 CmdEhto luokan metodi

```
<connectionStrings>
    <add name="Yhteinen" connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=C:\Recipes\Reffi_Yhteinen.mdb" providerName="System.Data.OleDb" />
    <add name="Linja1" connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=C:\Recipes\Reffi_Linja1.mdb" providerName="System.Data.OleDb" />
    <add name="Linja2" connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=C:\Recipes\Reffi_Linja1.mdb" providerName="System.Data.OleDb" />
</connectionStrings>
```

Ohjelma 26 Web.config tiedostossa määritellyt Connection Stringit